1. **知道Service吗，它有几种启动方式？**

这是我印象深刻的一道题，很明显它是我的第一次，那时我去一家公司（暂时叫它T公司吧）面试外派到韩国三星的工作机会。T公司的面试官是一个叫Bely架构师，显然那个时候Android开发是稀缺资源，知道Service那都不得了了，当然Bely也没打算为难我（必竟也工作4年多了，人长得也不错），我轻松对答：

Service是一个专门在后台处理长时间任务的Android组件，它没有UI。它有两种启动方式，startService和bindService。

你猜得没错，Bely紧接着问我：这两种启动方式的区别。

startService只是启动Service，启动它的组件（如Activity）和Service并没有关联，只有当Service调用stopSelf或者其他组件调用stopService服务才会终止。  
bindService方法启动Service，其他组件可以通过回调获取Service的代理对象和Service交互，而这两方也进行了绑定，当启动方销毁时，Service也会自动进行unBind操作，当发现所有绑定都进行了unBind时才会销毁Service。

这应该是比较关键的区别了，在面试前我刚刚用Serivce做过一个音乐播放器。几年后，我在深圳面试过很多人，他们中有60-70%的人没有使用Service的经验，让我一度感觉得深圳这座城市做Android开发的比较浮躁。因为这儿工作机会太多了，初级的开发者都比较急功近利，不需要在自己身上下太多的功夫也可以找到工作（当然这是片面的认识）。

当然还有其他的区别，如两种调用对Service生命周期函数影响，面试官也可以就这个问题展开一下。

当我遇到面试者知道怎么使用Service，也如多年前的我可以自如的答出startService和bindService的区别时，我一般会多问一句：

Service的onCreate回调函数可以做耗时的操作吗？

很多人都会说：可以。

原形毕露，他前面的回答只是在面试前预习了一下面试题而已。如果知道Service的onCreate是在主线程（ActivityThread）中调用的，耗时操作会阻塞UI，我一般再接着问：

如果需要做耗时的操作，你会怎么做？

问题便这样展开了，一个人是否真正懂得原理会灵活运用，一下子便能看出来。 当面试者回答到线程和Handler方式时，我会再问一下对方:

是否知道IntentService，在什么场景下使用IntentService？

这也是面试官要看的点，真正的项目需要一个开发人员对某个问题有一定的深度，也需要对整个Android的知识点有一定的广度。深度代表这个人对问题认真对待有钻研的精神，广度代表这个人在面对同一个问题时，会更容易从多种可行的方案中选出最合适的一种。

Service的实际项目中一直被很多人忽略，为什么我一再强调Service很重要，我们来看看，如果对Service完全无知会在工作中遇到什么问题。

场景：如果一个应用要从网络上下载MP3文件，并在Activity上展示进度条，这个Activity要求是可以转屏的。那么在转屏时Actvitiy会重启，如何保证下载的进度条能正确展示进度呢？

没有Service概念的人，一般想出来的方案如下：

在转屏前将进度缓存，转屏后再读出来。

使用android:configChanges设置，让转屏时Activity不销毁和重建。

针对第1个方案，我会继续问他将进度值存在哪里？ 转屏的过程中，我们知道Activity的重建算是比较耗时的，会可能会有几百毫秒以上，那么这时候下载线程仍然在工作，进度肯定和保存时的进度不一致了，如何处理这个问题呢？

第2个方案，大家可以自己展开思考，实际的项目中可能会需要额外做一些事情来处理ContentView的横竖布局的问题。

如果使用Service来解决这个问题，看似是比较完美的，不过就会涉及Activity（UI）和Service的交互问题，这个我们以后再讨论。

小结

当我们知道了Service的用途，心中有一个Service相关的概念时，针对实际的场景还是要做具体的分析再决定是否使用Service。因为Service仍然是在主线程中调用，还是要开线程才能处理长时间的工作，Service和UI的交互也让这个方式变得不那么简便。如果你只需要在当前界面去做一些耗时操作，界面退出或改变时，工作也要停止，那么这时直接使用Thread（或者AsyncTask, ThreadHandler）会更合适你。

## 用广播来更新UI界面好吗？

做为Android四大组件之一的，广播被很多人所熟知，可算是一种非常方便的解耦组件的手段。常用的方式是直接调用Context的接口（sendBroadcast & sendOrderBroadcast）发送两类型的广播：

Normal broadcasts无序广播，会异步的发送给所有的Receiver，接收到广播的顺序是不确定的，有可能是同时。  
Ordered broadcasts有序广播，广播会先发送给优先级高(android:priority)的Receiver，而且这个Receiver有权决定是继续发送到下一个Receiver或者是直接终止广播。

当然，静态和动态的两种注册Receiver的方式也难不住面试者。只是有时为了看一下面试者是否真的全面了解广播，会问一下：

除了上面的两种广播外，还有其他类型的广播吗？

允许我心里小小的邪恶一下。很少有人知道这种方式的，可以使用sendStickyBroadcast发送Sticky类型的广播。Sticky简单说就是，在发送广播时Reciever还没有被注册，但它注册后还是可以收到在它之前发送的那条广播。

有时候基于数据安全考虑，我们想发送广播只有自己（本进程）能接收到，那么该如何去做呢？

在我不知道有新的API或者框架时我常常喜欢用自己现有的知识去想方案，最后再Google一下看是否有更好的。这个问题，我会先想到权限，发送广播时限定有权限（receiverPermission）的接收者才能收到。但是我们知道APK太容易被反编译，注册广播的权限也只是一个字符串，并不安全。

然后可能使用Handler，没错，往主线程的消息池（Message Queue）发送消息，只有主线程的Handler可以分发处理它，广播发送的内容是一个Intent对象，我们可以直接用Message封装一下，留一个和sendBroadcast一样的接口。在handleMessage时把Intent对象传递给已注册的Receiver。

后来在看项目组的其他同事写代码时，发现还有一个LocalBroadcastManager类，查了一下官方文档是Support V4包里的一个类，其实现方式也是使用Handler，思路也是一样的。

BroadcastReceiver的生命周期

有些人并不态清楚Receiver也是有生命周期的，而且很短，当它的onReceive方法执行完成后，它的生命周期就结束了。这时BroadcastReceiver已经不处于active状态，被系统杀掉的概率极高，也就是说如果你在onReceive去开线程进行异步操作或者打开Dialog都有可能在没达到你要的结果时进程就被系统杀掉。因为这个进程可能只有这个Receiver这个组件在运行，当Receiver也执行完后就是一个empty进程，是最容易被系统杀掉的。替代的方案是用Notificaiton或者Service（这种情况当然不能用bindService）。

回到今天的面试题，使用广播来更新界面是否合适？

更新界面也分很多种情况，如果不是频繁地刷新，使用广播来做也是可以的。但对于较频繁地刷新动作，建议还是不要使用这种方式。广播的发送和接收是有一定的代价的，它的传输是通过Binder进程间通信机制来实现的（细心人会发现Intent是实现了Parcelable接口的），那么系统定会为了广播能顺利传递做一些进程间通信的准备。

除此之外，还可能有其他的因素让广播发送和到达是不准时的（或者说接收是会延时）。曾经看到有人在论坛上抱怨发几个广播都卡，Google的工程师是怎么混饭吃的。

这种情况可能吗？很可能，而且很容易发生。我们要先了解Android的ActivityManagerService有一个专门的消息队列来接收发送出来的广播，sendBroadcast执行完后就立即返回，但这时发送来的广播只是被放入到队列，并不一定马上被处理。当处理到当前广播时，又会把这个广播分发给注册的广播接收分发器ReceiverDispatcher，ReceiverDispatcher最后又把广播交给接Receiver所在的线程的消息队列去处理（就是你熟悉的UI线程的Message Queue）。

整个过程从发送--ActivityManagerService--ReceiverDispatcher进行了两次Binder进程间通信，最后还要交到UI的消息队列，如果基中有一个消息的处理阻塞了UI，当然也会延迟你的onReceive的执行。

小结

在项目里还是有遇到开发骨干也会在onReceive中开线程做耗时操作，很多时候他们这样做了并不会立刻就会产生问题，但是并不等于不会产生问题，当设备达到特定的临界条件时（如内存紧张），这些问题往往会在最终用户那里报出来。我们都有经验由用户报出来的随机BUG往往难于跟踪和修复，所以理解清楚一些基础的机制是很有必要的，它们能帮助我们避免一些隐藏的风险。

## 怎么理解Activity的生命周期？

这是一道几乎必问的Android面试题，当听到这题时大家的神经都会麻木，你问我背。说实话，如果你问我怎么理解，我也很难回答，因为这个问题太宽泛了，答什么都可以又答什么都难答好。有一次电话面试问了一个人，对方直接说不要问这种在网上到处都能看到的题，他只要上网一查就会。好吧，我脾气好，就接着问：  
“那问什么题呢？”  
“逻辑的，算法的，业务什么的，就是纯JAVA代码的。”  
“那么熟悉什么算法呢？”  
“。。。”（没答上）

但Activity的生命周期又至关重要，不问好像又对不起彼此。根据得到的反馈我后来稍稍改变了一下提问方式，不问怎么理解，而是给几个场景问怎么使用回调或回调的顺序（或要注意什么），从侧面看是否真正理解了。

下面还是先过一下官网上的Activity生命周期图，注意几个框线其实代表又可以细分为不同的周期。

错误!未指定文件名。

如果一个Activity在用户可见时才处理某个广播，不可见时注销掉，那么应该在哪两个生命周期的回调方法去注册和注销BroadcastReceiver呢？

Activity 的可见生命周期发生在 onStart调用与 onStop调用之间。在这段时间，用户可以在屏幕上看到 Activity 并与其交互。我们可以在 onStart中注册一个 BroadcastReceiver以监控影响 UI 的变化，并在用户无法再看到您显示的内容时在 onStop中将其取消注册。

如果对方回答是在onResume和onPause方法中，那么你可以去引导对方看看在这两个方法有什么不好的地方。

如果有一些数据在Activity跳转时（或者离开时）要保存到数据库，那么你认为是在onPause好还是在onStop执行这个操作好呢？

这题的要点和上一题是一样的，onPause较容易被触发，所以我们在做BroadcastReceiver注销时放在onStop要好些。onPause时Activity界面仍然是可见的，如弹出一个Dialog时。但在保存数据时，放在onPause去做可以保证数据存储的有效性，如果放在onStop去做，在某些情况下Activity走完onPause后有可能还没顺利走到onStop就被系统回收了。

但要注意在onPause中要非常迅速地执行完所需操作，不然会影响到下一个Activity的生命周期函数的调用。

简单说一下Activity A启动Activity B时，两个Activity生命周期的变化。

当一个 Activity 启动另一个 Activity 时，它们都会发生生命周期转变。第一个 Activity 暂停然后停止（但如果它在后台仍然可见，则不会停止，比如B是半透明的），系统会创建另一个 Activity。 如果这两个Activity 共用保存数据到文件或者数据库，必须要注意，在创建第二个 Activity 前，第一个 Activity 不会完全停止。更确切地说，启动第二个 Activity 的过程与停止第一个 Activity 的过程存在重叠。

以下是当 Activity A 启动 Activity B 时一系列操作的发生顺序：

Activity A 的 onPause方法执行。  
Activity B 的 onCreate、onStart和 onResume方法依次执行。  
然后，如果 Activity A 在屏幕上不再可见，则其 onStop方法执行。

您可以利用这种可预测的生命周期回调顺序管理从一个 Activity 到另一个 Activity 的信息转变。 例如，如果您必须在第一个 Activity 停止时向数据库写入数据，以便下一个 Activity 能够读取该数据，则应在 onPause而不是 onStop执行期间向数据库写入数据。

小结

这几个问题看似简单，其实还是很能问出面试者是否真正理解Activity生命周期的转变。经常有人会问，知道这些有什么用？又不能提升编程能力？编程能力是一个很复杂的体系，不能光看有技术含量或者高大上的算法才叫有能力，和悟道一样，我认为认识理清Android体系，顺应它的道（机制）开发应用也是一种能力。

## 如何判断Activity是否在运行？

如下这场景我相信很多人都遇到过，这段话也是从某个帖子截取出来的：

从Activity A 启动一个线程去进行网络上传操作，在A中设立一个回调函数，当上传操作完成以后，在A的这个回调函数中会弹出一个对话框，用来显示回调信息。可是当上传的过程还在进行的时候，我按下back键，A的activity 被销毁了，可是那个上传的线程还在进行，当那个线程结束后，本来应该在A中弹出一个对话框，可是由于A已经不存在了，系统就会报错提示，“将对话框显示在不存在的页面上”，然后程序就挂掉了。

我看到过很多人用Handler来充当上面所提到的“回调函数”，即工作线程完成工作后，通过主线程的Handler处理UI更新，如弹出提示Dialog。可能有些人没有弄明白，Activity都被系统销毁了，工作线程怎么还能调它的变量呢？其实所谓的Activity销毁只是不再受系统的AMS控制，但Activity这个对象的实例还是存在于内存中的，具体什么时候真正把这个对象实例也销毁（回收）了，就要看内存回收机制了，哪怕是这个实例没有可达的引用了也不一定会马上回收。

针对这种用Handler更新UI的情况，我们需要在操作UI前判断一下此Activity是否已被销毁。很多人可能都用过isFinishing来判断，用多了就会发现好象不太准，为什么呢，看一下它的源代码：

/\*\*

\* Check to see whether this activity is in the process of finishing,

\* either because you called {@link #finish} on it or someone else

\* has requested that it finished. This is often used in

\* {@link #onPause} to determine whether the activity is simply pausing or

\* completely finishing.

\*

\* @return If the activity is finishing, returns true; else returns false.

\*

\* @see #finish

\*/

public boolean isFinishing() {

return mFinished;

}

/\*\*

\* Returns true if the final {@link #onDestroy()} call has been made

\* on the Activity, so this instance is now dead.

\*/

public boolean isDestroyed() {

return mDestroyed;

}

而mFinished是在finish()中被赋值的，也就是说只有通过调用finish()结束的Activity，mFinished的值才会被置为true。所以有时候Activity的生命周期没有按我们预想的来走时（如内存紧张时），会出现判断出错的情况。

Android源码可以使用这个网站查询：[androidxref](https://link.jianshu.com?t=http:/androidxref.com/)

看看Google工程师是怎么判断的（来源于Android源码中的Call应用，AsyncTask中的onPostExecute片段）：

@Override

protected void onPostExecute(Void result) {

final Activity activity = progressDialog.getOwnerActivity();

if (activity == null || activity.isDestroyed() || activity.isFinishing()) {

return;

}

if (progressDialog != null && progressDialog.isShowing()) {

progressDialog.dismiss();

}

}

多了一个isDestroyed()的判断。

小结

如果对方没听说过isFinishing函数，那可以让他从自己的角度看如何解决这个问题，正好可以看看他的逻辑思维是否清晰合理。工作中往往会遇到，一些求职者由于之前是做其他方面刚转Android开发，对Android的了解还不够，但有很强理解和学习能力，通过引导发现他可以快速的得到合理的解决方案的话，我一般都很乐意要这样的人。

## 自定义View的状态是如何保存的？

Android有一套标准的做法，做过自定义View的人很容易遇到这个问题，因为Activity转屏，或Home键到后台很容易在被系统销毁，恢复时我们肯定是希望看到View保持之前状态。

提示：系统内存紧张时会主动销毁这类Activity，做为开发我们可以利用DDMS的stop process模拟这一动作而不必等到内存紧张时，如下图所示：

红色的STOP图标可以杀掉当前选择的进程

这个标准的做法可以随便从一个Android自带的控件中看到，如TextView的源代码([SavedState extends BaseSavedState](https://link.jianshu.com?t=http:/androidxref.com/4.4.2_r2/xref/frameworks/base/core/java/android/widget/TextView.java))。

/\*\*

\* User interface state that is stored by TextView for implementing

\* {@link View#onSaveInstanceState}.

\*/

public static class SavedState extends BaseSavedState {

int selStart;

int selEnd;

CharSequence text;

boolean frozenWithFocus;

CharSequence error;

SavedState(Parcelable superState) {

super(superState);

}

@Override

public void writeToParcel(Parcel out, int flags) {

super.writeToParcel(out, flags);

out.writeInt(selStart);

out.writeInt(selEnd);

out.writeInt(frozenWithFocus ? 1 : 0);

......

BaseSavedState是View的一个内部静态类，从代码上我们也很容易看出是把控件的属性（如selStart）打包到Parcel容器，Activity的onSaveInstanceState、onRestoreInstanceState最终也会调用到控件的这两个同名方法。

Parcel相关的问题以后专门再讲，现在我们可以焦点先放在状态保存上，先看一下Activity的状态如何保存的：

Activity 状态

注：无法保证系统会在销毁Activity前一定调用onSaveInstanceState，例如用户使用“返回” 按退出 Activity 时，因为用户的行为是在显式关闭 Activity，所以不会调用onSaveInstanceState。

如果系统调用onSaveInstanceState，那么它是在onStop还是在onPause之前执行呢？

可以肯定的是它会在调用 onStop之前，但是是不是在onPuase之前就不能确认了，要看情况，官方文档在说明这个执行顺序时用了“可能”这个词。

小结

Activity类的onSaveInstanceState默认实现会恢复Activity的状态，默认实现会为布局中的每个View调用相应的 onSaveInstanceState方法，让每个View都能保存自身的信息。

这里需要注意一个细节：想要保存View的状态，需要在XML布局文件中提供一个唯一的ID（android:id），如果没有设置这个ID的话，View控件的onSaveInstanceState是不会被调用的。

1. **通过new创建的View实例它的onSaveStateInstance会被调用吗？**

这里再强调一下，自定义View控件的状态被保存需要满足两个条件：

View有唯一的ID；

View的初始化时要调用setSaveEnabled(true) ；

简单看一下View状态保存和读取的调用过程：

里面的SparseArray(完整的参数是：SparseArray<Parcelable> )是一个KEY-VALUE的Map，KEY当然就是View的ID了。所以细看一下源码的调用过程，你一下就理解为什么一定要给View调置一个唯一的ID了。

那好，现在回答上这个网友的问题“通过new创建的View实例它的onSaveStateInstance会被调用吗？”，答案还是一样的如果设置了ID就会。其实我们在XML文件中配置的布局和属性最终都是通过LayoutInflater中的inflate方法去加载，由它去创建各个View的实例（还是用new），并根据XML文件中的属性设置相关的值。

我们再展开一下，如果我们定义了一个自义的Layout，在同一个界面中引用了两次这个自定义的Layout（如下图的myLayout1 & myLayout2），那么它的状态会发生什么情况呢？

我们会发现两个Layout的状态被保存了，但Layout中的每个子View状态是相同的，被最后一个保存的View覆盖了。这也是为什么我们要强调View的ID要唯一的原因。

如何解决这个问题呢？留给大家去展开吧，你是面试官的话，也可以留给来面试的人，不过这里太细节了建议最好做为附加题让对方试试（答不上不扣分）。

## Java的值传递和引用传递问题

看过一些Java教材的话，对这句话应该不陌生：

“在Java里面参数传递都是按值传递”  
这句话的意思是：按值传递是传递的值的拷贝，按引用传递其实传递的是引用的地址值，所以统称按值传递。

简单的说，基本类型是按值传递的，方法的实参是一个原值的复本。类对象是按对象的引用地址（内存地址）传递地址的值，那么在方法内对这个对象进行修改是会直接反应在原对象上的（或者说这两个引用指向同一内存地址）。不过要注意String这个类型，如下代码：

public static void main(String[] args) {

String x = new String("goeasyway");

change(x);

System.out.println(x);

}

public static void change(String x) {

x = "even";

}

可以自己思考一下会输出什么结果，为什么。String的对象是不可修改的，如果真正要达到改变字符串的效果，我们需要使用StringBuilder来替代String。

结论

很遗憾，我在实际中发现很多人都不会去重视基础问题，保括一些资深的开发也不太清楚Java还有这两种区分。在之前我呆过的一家公司，有一位女程序员直接在代码里使用一个未经声明的方法，并给这个方法传参数，然后来问我为什么编译不过呢。但这却不妨碍她仍然可以为客户解决问题，可以获得组内的最佳员工评选。所以有很多人并不重视基础的问题，总认为不知道也无防，用的时候有问题自然会报出来，到时候再解决就好了，你知道的也没比我多能耐。我只能说，知道的话确实不比别人多能耐，只是多了一份从容。

## 能讲讲Android的Handler机制吗？

Android应用程序是通过消息来驱动的，系统为每一个应用程序维护一个消息队例（MessageQueue），应用程序的主线程不断地从这个消息队例中获取消息（Looper），然后对这些消息进行处理（Handler），这样就实现了通过消息来驱动应用程序的执行。

Handler在整个Android应用中占有很重要的地方，所以面试时我们经常要考查一下面试者是否了理解它的原理，并且能够将大体的流程表述清楚，沟通和表达能力有时比技术能力更重要。

要讲清楚Android中的消息机制，肯定要先表述一下和Handler相关的一些类：

Message：消息分为硬件产生的消息(如按钮、触摸)和软件生成的消息；

MessageQueue：消息队列的主要功能向消息池投递消息(MessageQueue.enqueueMessage)和取走消息池的消息(MessageQueue.next)；

Handler：消息辅助类，主要功能向消息池发送各种消息事件(Handler.sendMessage)和处理相应消息事件(Handler.handleMessage)；

Looper：不断循环执行(Looper.loop)，按分发机制将消息分发给目标处理者。

Handler相关类的代码量并不大，建议大家都去看一下，网上也有很多介绍和分析这些源码的文章，大家自己Google一下。大家把代码过了一遍后，会更加深对整个过程的理解，讲起来就从容多了。不建议大家为了面试去背书。

面试时，如果一个人可以清楚的表达Handler的运行机制，那么我们接下来会主要问一下一些实际开发中注意的地方。比如会问在一个工作线程中创建自己的消息队例应该怎么做？

其实就是想从侧面验证他是否正的了解，是否知道要调用Looper.prepare（在每个线程只允许执行一次）。

或者再问问是否用过HandlerThread，它有什么优缺点等。

注意：Handler可能会引起的内存泄露

在Activity中像这样创建一个Handler再正常不过了。

private final Handler handler = new Handler() {

@Override

public void handleMessage(Message msg) {

}

};

但是，其实上面的代码可能导致内存泄露，当你使用Android lint工具的话，会得到这样的警告:

In Android, Handler classes should be static or leaks might occur, Messages enqueued on the application thread’s MessageQueue also retain their target Handler. If the Handler is an inner class, its outer class will be retained as well. To avoid leaking the outer class, declare the Handler as a static nested class with a WeakReference to its outer class

有兴趣的可以细看一下这篇译文：《Android中Handler引起的内存泄露》

结论

诸如Handler这类耳熟能详的概念，但其实用起来又不复杂，面试时一般会更在意对方的表达上，看对方是否能用语言有效的组织语句。最后针对一个问题，还是要用一点小的细节验证对方是否正的做过。有些网友可能会觉得是被故意刁难，但如果面试官只提出一个问题，你说了答案后他就嗯一下就立即问下一主题，没有和你就这个问题再扩展一下，你是否也会觉得他什么都不懂，也会质疑他能否辨别出面试者的真实水平？

## 两个Activity之间如何传递参数？

在Android应用中，Activity占有极其重要的地位，Activity间的跳转更是加常便饭。即然跳转（界面切换）不可避免，那么在两个Activity之间传递参数就是一个常见的需求。大多数时候，我们也就传递一些简单的int，String类型的数据，实际中也有看到传递List和Bitmap的。

那么我们先回答这个题，如何传递参数：

使用Intent的Bundle协带参数，就是我们常用的Intent.putExtra方法。

做为面试官，紧接着可以问：除了传递基本类型外，如何传递自定义的对象呢？

这个问题就是想引出Android的Parcelable。一般很多面试者都有用过传递实现了Serializable接口的自定义对象的经验，因为这个很简单，加句代码就搞定了。而Parcelable的实现要多一些代码，典型的写法如下：

public class MyParcelable implements Parcelable {

private int mData;

public int describeContents() {

return 0;

}

public void writeToParcel(Parcel out, int flags) {

out.writeInt(mData);

}

public static final Parcelable.Creator<MyParcelable> CREATOR

= new Parcelable.Creator<MyParcelable>() {

public MyParcelable createFromParcel(Parcel in) {

return new MyParcelable(in);

}

public MyParcelable[] newArray(int size) {

return new MyParcelable[size];

}

};

private MyParcelable(Parcel in) {

mData = in.readInt();

}

}

那我们为什么要考察对方会不会用Parcelable呢？先看一下这Parcelable和Serializable的区别：

Serializalbe会使用反射，序列化和反序列化过程需要大量I/O操作，Parcelable自已实现封送和解封（marshalled &unmarshalled）操作不需要用反射，数据也存放在Native内存中，效率要快很多。

有人比较过它们两个的效率差别：

不同类型的数据不一定差据这么大，但却很直观的展示了Pacelable比Serializable高效。

有时面试官还可以追问一下：Parcelable和Parcle这两者之间的关系。

Parcelable 接口定义在封送/解封送过程中混合和分解对象的契约。Parcelable接口的底层是Parcel容器对象。Parcel类是一种最快的序列化/反序列化机制，专为Android中的进程间通信而设计。该类提供了一些方法来将成员容纳到容器中，以及从容器展开成员。

现在我们知道了如何传递自定义的对象，那么在两个Activity之前传递对象还要注意什么呢？

一定要要注意对象的大小，Intent中的Bundle是在使用Binder机制进行数据传递的，能使用的Binder的缓冲区是有大小限制的（有些手机是2M），而一个进程默认有16个binder线程，所以一个线程能占用的缓冲区就更小了（以前做过测试，大约一个线程可以占用128KB）。所以当你看到“The Binder transaction failed because it was too large.”这类TransactionTooLargeException异常时，你应该知道怎么解决了。

因此，使用Intent在Activity之间传递List和Bitmap对象是有风险的。

面试官可以就这个问题再展开，看面试者如何解决。

还有一个要注意的：因为android不同版本Parcelable可能不同，所以不推荐使用Parcelable进行数据持久化。之前我有过一次，将Android的PackageInfo进行持久化到数据库，结果用户升级Android系统后，再从数据库解封PackageInfo时应用就Crash了。

结论

对于初级的程序员来说，只要能抓住老鼠，白猫或者黑猫甚至是小狗都是没有区别的。但对于应用的流畅和体验来说，100毫秒和1000毫秒是有很大区别的。很多程序员眼里无关紧要的差别，最终在用户那儿会被几倍十几倍的放大，老板也会因为用户的投述而斥责你。因为总会有用户在用性能很差的手机，总有用户手机的使用情况很复杂（内存紧张，网络复杂等等），总有用户本人就很奇葩不会按你指定的套路出拳！当你鄙视老板不懂代码的艺术时，老板也会鄙视你不懂用户不懂细节的重要性，活该你一辈子做程序员。

所以，在能使用的Parcelable的地方，请不要贪图简便直接Serializable，实在懒的话也可以试试用插件自动生成Pracelabel的模板代码：android-parcelable-intellij-plugin

## 如何理解Android中的Context，它有什么用？

官方文档对于Context的解释：

Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

中文翻译“Context”为“上下文，背景，环境，语境”，有点抽象。对照上面的英文，也很容易理解它的意思：Context提供了一个应用的运行环境，通过这个上下文应用才可以访问资源，才能完成和其他组件、服务的交互。它就是一个调用者和具体实现的桥接。

再看看这个图，可以直观的了解Context相关类的继承关系：

图片来自网络

具体的实现，大家可以看一看ContextImpl这个类的源码，理一下它和LoadApk的关系，大家就会很容易明白Context是如何处理资源的。

注意：上图中的mPackageInfo是一个LoadApk对像，这个LoadApk是一个hide类型的类，所以你在SDK中是找不到它的。

但是，面对天天见的“大众脸同学”Context，很多人更想知道，理解它有什么实际的意义。做应用开发，我们只要会使用它不就行了吗？

我先问一个简单的问题：

Application（或者Service）和Activity都可以调用Context的startActivity方法，那么在这两个地方调用startActivity有区别吗？

如果你曾经遇到过，就会知道在Application（或者Service）需要给Intent设置Intent.FLAG\_ACTIVITY\_NEW\_TASK才能正常启动Activity，这就会引出Activity的Task栈问题，以后再做分析。

理解Context，对于我经历的项目来说，最有用的就是对于插件框架的开发了。如果有面试官问你:

Context的实例是什么时候创建的？一个应用里面会有几个Context的实例？

对于一般的应用来说，你会觉得这两个问题很无聊。但如果你需要做插件开发，上面的问题就变成是很关键的问题了。你的插件框架会是一个小型的Android Framework层，你当然得自己处理插件的Application和Activity创建，那么你肯定要解决好这两个问题。详情可以查看ActivityThread这个类的源码。

也就是说，如果我要招一个做插件开发的人，我肯定要了解一下他对这些问题的看法。或者一个人的简历上说他对Android Framework层很精通，也可以拿这些问题检测一下。

小结

有网友问“面试官是怎么考虑求职者的经验、学历、编程水平”这些方面的，其实这个问题不能脱离实际的公司和项目来回答。我只能说几个场景，有些公司有人才培养计划项目也不紧张，那么他们在招人时是以培养和贮备为目的，会更重视面试者的理解和学习能力。但如何一家公司急切需要人进来解决问题，他们就会更在乎你的项目经验了，最好是直接招以前就做过类似项目的。如果一家公司只是需要码农来搬代码，那么只要不是太差的，他们会更看重性价比。

所以有公司关注这些对Context或Framework方面的理解的面式题，一是他们应用可能遇到了一些问题，需要一些对机制比较了解的开发来解决；二是想通过这类问题，考察面试者是否真如简历上般资深，因为他们相信做多了项目的人，很容易遇到机制方面的问题（如Dex的65535方法数限制）。

作者：goeasyway

链接：https://www.jianshu.com/p/0754e65a5744

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 如何优化ListView的性能？

在回答这个问题前，我认为很有必要和大家讲几点和getView相关的问题。我们设置或者优化ListView的性能很多时候都是在getView中完成的，反过来说就是很多性能问题都是由于没有正确使用getView造成的。

public View getView(int position, View convertView, ViewGroup parent)

所以我们不妨先思考一下如下的几个问题：

在一次显示ListView的界面时，getView会被执行几次？

每次getView执行时间应该控制在多少毫秒之内？

getView中设置listener要注意什么？

首先我们要知道ListView的ItemView有一个复用机制，简单看如下图所示，ListView中有一个RecycleBin类复负回收不可见且可能被再次使用的ItemView，由ScrapView存储。

所以我在们设置Listener进就要注意，使用convertView时需要重新设置一个Listener，保括一些数据也需要重设置，不然可能会显示之前那个ItemView在回收前的状态。

在绘制ListView前往往要计算它的高度，所以一个ListView界面上可以看到6个ItemView，但是getView的执行次数却有可能是12次，多出的次数用来计算高度（这个可以通过设置ListView的height为0来避免）。所以要避免在getView中进行逻辑运算，两次计算同一逻辑完全是浪费。

每个getView的执行时间更是少得可怜，很多人可能对这个时间没有概念，我可以简单的给大算一下：

1秒之内屏幕可以完成30帧的绘制，人才能看到它比较流畅（苹果是接近60帧，高于60之后人眼也无法分辨）。

每帧可使用的时间：1000ms/30 = 33.33 ms

每个ListView一般要显示6个ListItem，加上1个重用convertView：33.33ms/7 = 4.76ms

即是说，每个getView要在4.76ms内完成工作才会较流畅，但是事实上，每个getView间的调用也会有一定的间隔（有可能是由于handler在处理别的消息），UI的handler处理不好的话，这个间隔也可难会很大（0ms-200ms）。结论就是，留给getView使用的时间应该在4ms之内，如果不能控制在这之内的话，ListView的滑动就会有卡顿的现象。

了解了这几个问题，现在我们回来这次主要考查的面试题上，如何进行ListView的性能优化，让它滑动更加流畅。大家一般常用如下方法：

重用ConvertView;

使用View Holder模式；

使用异步线程加载图片（一般都是直接使用图片库加载，如Glide, Picasso）；

我认为这些是面试者必备的知识点，如果连这些都说不清楚的话，也没有必要再深入问了。针对面试者的回答，可以适当选一两点追问一下，看是否真正明白。如：ViewHolder为什么能够起到优化性能的作用？

除此之前还有一些优化建议：

在adapter的getView方法中尽可能的减少逻辑判断，特别是耗时的判断；

避免GC（可以从LOGCAT查看有无GC的LOG）；

在快速滑动时不要加载图片；

将ListView的scrollingCache和animateCache这两个属性设置为false（默认是true）;

尽可能减少List Item的Layout层次（如可以使用RelativeLayout替换LinearLayout，或使用自定的View代替组合嵌套使用的Layout）；

关于第4点，发现在一些型号的手机（如华为的P7）上特别管用，当其也优化都做完之后，有无这两项设置滑动的卡顿情况有明显不同。

<Listview

android:scrollingCache="false"

android:animationCache="false"

小结

关于ListView有很多方面可以考察面试者，因为它实在是用的太频繁了，双方都能对某个问题点进行展开。如果一个面试者都没有做过ListView优化，那么如果不是他写的代码太少就是他使用ListView加载的数据太简单（可能只有几十项），其本上没有其他选项。所以这一题是很能看出面试者的项目经验和实际的开发水平，属于面试必考题之一。

作者：goeasyway

链接：https://www.jianshu.com/p/8fd5fa90ee6c

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 如何实现应用内多语言切换？

我们知道Android的多语言实现很简单，可以在不同的语言环境下使用不同的资源。在不同的res/value-xx下放置不同语言的strings.xml实现字符的本地化，而这个value-xx目录的选择是根据Resource中的Configuration.Locale这项的值来决定的。如zh中文，就会选择value-zh目录，如果没有匹配到（即APK中没有value-zh目录）就使用默认的value目录中的字符资源。

其实最终实现字符串的选择都是在Assets这个类中，通过Native的方法来加载相应的字符串资源。

然而，我们还是会有一些业务场景需要不根据Android系统的Locale配置就改变应用的语言。实现的方式也很简单，直接调用Android开放的接口Resources.updateConfiguration：

public static void changeSystemLanguage(Context context, String language) {

if (context == null || TextUtils.isEmpty(language)) {

return;

}

Resources resources = context.getResources();

Configuration config = resources.getConfiguration();

if (Locale.SIMPLIFIED\_CHINESE.getLanguage().equals(language)) {

config.locale = Locale.SIMPLIFIED\_CHINESE;

} else {

config.locale = new Locale(language);

}

resources.updateConfiguration(config, null);

}

上面的代码，我们可以在应用内通过language的值指定是显示哪种语言，当然language值我们需要保存在Preferences或者数据库中。

好像很简单，我们的项目为什么还会出现问题呢？而且大家都不知道如何下手，因为在Android N之前的版本都是可以正常切换语言的。后来我跟了一下，发现在MainActivity和SplashActivity这些Activity有继承了自已扩展的BaseActivity，而这个BaseActivity有这样一段代码：

@Override

public Resources getResources() {

Resources res = super.getResources();

Configuration config = new Configuration();

config.setToDefaults();

res.updateConfiguration(config, res.getDisplayMetrics());

return res;

}

config.setToDefaults会将Locale的值设为null，而再调用updateConfiguration可能会根据Android系统的语言重新设置Resources中的Locale。好吧，只是假设，还没有看到Android N的源代码。不过去掉这段代码后，在Android N（Preview）手机上切换语言正常了。

小结

今天遇到的问题，是以前遗留的代码埋下的坑终于暴露出来，也是这个项目缺乏代码审查（Code Review）机制的结果。找了几个人也无法说清覆写getResources这个方法的用意，最终也只能按历史问题处理了，是历史总有一些说不清楚的事，对吧。

回到这个面试题，现在你知道了可以在应用内切换语言（当然也可以修改Configuration的其他值），那么你有没有想过，如果不知道这个updateConfiguration的存在，你会怎么实现这个需求呢？或者说没有人和你说过updateConfiguration，你能找到它吗？

作者：goeasyway

链接：https://www.jianshu.com/p/024f46834485

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 在项目中使用AsyncTask会有什么问题吗？

那我们考查AsyncTask会问些什么呢？得先问问会不会用吧，看看知不知道有onProgressUpdate方法。

其次问一下是怎么理解AsyncTask的机制，有没有看过它的源代码？

这个问题主要看对方是否对Android的东西有好奇心，会主动去看AsyncTask的源码，或者能大体地讲清楚AsyncTask的原理。一般有好奇心的同学都比较善长学习，善长学习的人都能比较快融入团队。

AnsycTask执行任务时，内部会创建一个进程作用域的线程池来管理要运行的任务，也就就是说当你调用了AsyncTask.execute()后，AsyncTask会把任务交给线程池，由线程池来管理创建Thread和运行Therad。最后和UI打交道就交给Handler去处理了。

我们在实际的项目中，还需要关注一些问题：

线程池可以同时执行多少个TASK？

Android 3.0之前（1.6之前的版本不再关注）规定线程池的核心线程数为5个（corePoolSize），线程池总大小为128（maximumPoolSize），还有一个缓冲队列（sWorkQueue，缓冲队列可以放10个任务），当我们尝试去添加第139个任务时，程序就会崩溃。当线程池中的数量大于corePoolSize，缓冲队列已满，并且线程池中的数量小于maximumPoolSize，将会创建新的线程来处理被添加的任务。如下图会出现第16个Task比第6－15个Task先执行的情况。

多个AsyncTask任务是串行还是并行？

从Android 1.6到2.3(Gingerbread) AsyncTask是并行的，即上面我们提到的有5个核心线程的线程池（ThreadPoolExecutor）负责调度任务。从Android 3.0开始，Android团队又把AsyncTask改成了串行，默认的Executor被指定为SERIAL\_EXECUTOR。

/\*\*

\* An {@link Executor} that executes tasks one at a time in serial

\* order. This serialization is global to a particular process.

\*/

public static final Executor SERIAL\_EXECUTOR = new SerialExecutor();

从它的说明也可以看出是串行的。如需要并行，可以通过设置executeOnExecutor(Executor)来实现多个AsyncTask并行。

public final AsyncTask<Params, Progress, Result> executeOnExecutor(Executor exec,

Params... params) {

AsyncTask容易引发的Activity内存泄露

如果AsyncTask被声明为Activity的非静态的内部类，那么AsyncTask会保留一个对创建了AsyncTask的Activity的引用。如果Activity已经被销毁，AsyncTask的后台线程还在执行，它将继续在内存里保留这个引用，导致Activity无法被回收，引起内存泄露。

当然，最后少不了问一句：“你在项目中，会用什么方案来替换AsyncTask呢？”

小结

感觉对AsyncTask的使用有点“成也萧何败萧何”的味道，它简单的解决了UI和后台线程交互的问题，但如果忽视它的限制（缺陷）和各版本不一致的线程池方式，可能会达不到预想的效果。最后发现没有使用过AsyncTask会被鄙视，如果你在实际项目中使用了AsyncTask也会被鄙视。不过，学习它对理解Android的机制和线程池的使用还是很的意义的，所以强烈建议大家读一下它的源码。

作者：goeasyway

链接：https://www.jianshu.com/p/c925b3ea1444

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 修改SharedPreferences后两种提交方式有什么区别？

SharedPreferences类是一个接口类，真正的实现类是SharedPreferencesImpl。修改SharedPreferences需要获取它的Editor，在对Editor进行put操作后，最后通过commit或者apply提交修改到内存和文件。当然有了两种都可以提交的方法，肯定要区别一下的。从实现类SharedPreferencesImpl的源码上看也很容易看出两者的区别：

commit这种方式很常用，在比较早的SDK版本中就有了，这种提交修改的方式是同步的，会阻塞调用它的线程，并且这个方法会返回boolean值告知保存是否成功（如果不成功，可以做一些补救措施）。

而apply是异步的提交方式，目前Android Studio也会提示大家使用这种方式。

还有一点用得比较少的，就是SharedPreferences还提供一个监听接口可以监听SharedPreferences的键值变化，需要监控键值变化的可以用registerOnSharedPreferenceChangeListener添加监听器。

public interface SharedPreferences {

/\*\*

\* Interface definition for a callback to be invoked when a shared

\* preference is changed.

\*/

public interface OnSharedPreferenceChangeListener {

void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key);

}

多进程操作和读取SharedPreferences的问题

前段时间，项目组里发现一个偶现的问题，从Http明明获取了正确的数据保存到SharedPreferences，但立即再从SharedPreferences读取这个值时发现是初始值。开始大家一直把精力放在Http的请求上，最后才发现是SharedPreferences多进程间数据共享会导致的问题。

在SDK 3.0及以上版本，可以通过Context.MODE\_MULTI\_PROCESS属性来实现SharedPreferences多进程共享。如下设置：

public static SharedPreferences getSharedPreferences(String name) {

if (null != context) {

if (Build.VERSION.SDK\_INT >= 11) {

return context.getSharedPreferences(name, Context.MODE\_MULTI\_PROCESS);

} else {

return context.getSharedPreferences(name, Context.MODE\_PRIVATE);

}

}

return null;

}

本来以为通过MODE\_MULTI\_PROCESS属性使用SharedPreferences就可以实现不同时程间共享数据，但是在真正使用中确发现有会有一定概率出现这个取值出错（变为初始值）问题。

最后发现在官网上Google也在SDK 6.0的版本将这个MODE\_MULTI\_PROCESS标识为deprecated（不赞成使用）。目前来说，越来越多的项目在不断的膨胀，为了降低单个进程的内存占用率，使用"android:process"配置一些组件在单独的进程中运行已经是司空见惯了，所以大家在遇到自己的项目有多进程时，要注意一下SharedPreferences的问题。

小结

在一个进程中，SharedPreference往往建单个实例就可以了，一般不会出现并发冲突，如果对提交的结果不关心的话，建议使用apply，当然需要确保提交成功且有后续操作的话，还是需要用commit的。

因为SharedPreferences在多进程方面的问题，大家也可以思考下能不能自己实现一个加强版的SharedPreferences解决这些问题，网上也有一些开源的替代方案，如Github上的tray。（建议大家先想一下，再看这个项目。）

作者：goeasyway

链接：https://www.jianshu.com/p/4dd53e1be5ba

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 有使用过ContentProvider码？能说说Android为什么要设计ContentProvider这个组件吗？

ContentProvider应用程序间非常通用的共享数据的一种方式，也是Android官方推荐的方式。Android中许多系统应用都使用该方式实现数据共享，比如通讯录、短信等。但我遇到很多做Android开发的人都不怎么使用它，觉得直接读取数据库会更简单方便。

那么Android搞一个内容提供者在数据和应用之间，只是为了装高大上，故弄玄虚？我认为其设计用意在于：

封装。对数据进行封装，提供统一的接口，使用者完全不必关心这些数据是在DB，XML、Preferences或者网络请求来的。当项目需求要改变数据来源时，使用我们的地方完全不需要修改。

提供一种跨进程数据共享的方式。

应用程序间的数据共享还有另外的一个重要话题，就是数据更新通知机制了。因为数据是在多个应用程序中共享的，当其中一个应用程序改变了这些共享数据的时候，它有责任通知其它应用程序，让它们知道共享数据被修改了，这样它们就可以作相应的处理。

ContentResolver接口的notifyChange函数来通知那些注册了监控特定URI的ContentObserver对象，使得它们可以相应地执行一些处理。ContentObserver可以通过registerContentObserver进行注册。

既然是对外提供数据共享，那么如何限制对方的使用呢？

android:exported属性非常重要。这个属性用于指示该服务是否能够被其他应用程序组件调用或跟它交互。如果设置为true，则能够被调用或交互，否则不能。设置为false时，只有同一个应用程序的组件或带有相同用户ID的应用程序才能启动或绑定该服务。

对于需要开放的组件应设置合理的权限，如果只需要对同一个签名的其它应用开放content provider，则可以设置signature级别的权限。大家可以参考一下系统自带应用的代码，自定义了signature级别的permission：

<permission android:name="com.android.gallery3d.filtershow.permission.READ"

android:protectionLevel="signature" />

<permission android:name="com.android.gallery3d.filtershow.permission.WRITE"

android:protectionLevel="signature" />

<provider

android:name="com.android.gallery3d.filtershow.provider.SharedImageProvider"

android:authorities="com.android.gallery3d.filtershow.provider.SharedImageProvider"

android:grantUriPermissions="true"

android:readPermission="com.android.gallery3d.filtershow.permission.READ"

android:writePermission="com.android.gallery3d.filtershow.permission.WRITE" />

如果我们只需要开放部份的URI给其他的应用访问呢？可以参考Provider的URI权限设置，只允许访问部份URI，可以参考原生ContactsProvider2的相关代码（注意path-permission这个选项）：

<provider android:name="ContactsProvider2"

android:authorities="contacts;com.android.contacts"

android:label="@string/provider\_label"

android:multiprocess="false"

android:exported="true"

android:grantUriPermissions="true"

android:readPermission="android.permission.READ\_CONTACTS"

android:writePermission="android.permission.WRITE\_CONTACTS">

<path-permission

android:pathPrefix="/search\_suggest\_query"

android:readPermission="android.permission.GLOBAL\_SEARCH" />

<path-permission

android:pathPrefix="/search\_suggest\_shortcut"

android:readPermission="android.permission.GLOBAL\_SEARCH" />

<path-permission

android:pathPattern="/contacts/.\*/photo"

android:readPermission="android.permission.GLOBAL\_SEARCH" />

<grant-uri-permission android:pathPattern=".\*" />

</provider>

ContentProvider接口方法运行在哪个线程中呢？

ContentProvider可以在AndroidManifest.xml中配置一个叫做android:multiprocess的属性，默认值是false，表示ContentProvider是单例的，无论哪个客户端应用的访问都将是同一个ContentProvider对象，如果设为true，系统会为每一个访问该ContentProvider的进程创建一个实例。

这点还是比较好理解的，那如果我要问每个ContentProvider的操作是在哪个线程中运行的呢（其实我们关心的是UI线程和工作线程）？比如我们在UI线程调用getContentResolver().query查询数据，而当数据量很大时（或者需要进行较长时间的计算）会不会阻塞UI线程呢？

要分两种情况回答这个问题：

ContentProvider和调用者在同一个进程，ContentProvider的方法（query/insert/update/delete等）和调用者在同一线程中；

ContentProvider和调用者在不同的进程，ContentProvider的方法会运行在它自身所在进程的一个Binder线程中。

但是，注意这两种方式在ContentProvider的方法没有执行完成前都会blocked调用者。所以你应该知道这个上面这个问题的答案了吧。

也可以看看CursorLoader这个类的源码，看Google自己是怎么使用getContentResolver().query的。

ContentProvider是如何在不同应用程序之间传输数据的？

这个问题点有些深入，大家要对Binder进程间通信机制比较了解。因为之前我们有提到过一个应用进程有16个Binder线程去和远程服务进行交互，而每个线程可占用的缓存空间是128KB这样，超过会报异常。ContentResolver虽然是通过Binder进程间通信机制打通了应用程序之间共享数据的通道，但Content Provider组件在不同应用程序之间传输数据是基于匿名共享内存机制来实现的。有兴趣的可以查看一下老罗的文章Android系统匿名共享内存Ashmem（Anonymous Shared Memory）简要介绍和学习计划。

用个图总结一下

小结

初学开发或者开发经验少的人，往往不喜欢在写代码时思考如何应对以后可能产生的变化和相关接口的扩展能力，所以他们往往看不到ContentProvider的巨大好处。我觉得一般常见到的组件或概念，大家不一定需要每个都去偿试写写代码Demo（当然这样最好），但我认为对它们仍然需要有足够的重视，要去理清它的计设思路和使用场景。这样对你是极好的，之后你在面对问题时才会有多种解决方案的选择机会。

人生，比的不就是谁有更多的选择吗？

作者：goeasyway

链接：https://www.jianshu.com/p/380231307070

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 如何处理线程同步的问题？

有可能很多人对插件并不了解，不过没关系，这个需求简单地说就是主线程要等待多个子线程全部完成工作后，才能继续执行。

说到多线程的同步问题，面试多的人应该很容易被面试官问：Object的wait和notify/notifyAll如何实现线程同步？

在Object.java中，定义了wait(), notify()和notifyAll()等接口。wait()的作用是让当前线程进入等待状态，同时，wait()也会让当前线程释放它所持有的锁。而notify()和notifyAll()的作用，则是唤醒当前对象上的等待线程；notify()是唤醒单个线程，而notifyAll()是唤醒所有的线程。

wait和yield（或sleep）的区别？

wait()是让线程由“运行状态”进入到“等待(阻塞)状态”，而yield()是让线程由“运行状态”进入到“就绪状态”，从而让其它具有相同优先级的等待线程获取执行权；但是，并不能保证在当前线程调用yield()之后，其它具有相同优先级的线程就一定能获得执行权。

wait()是会线程释放它所持有对象的同步锁，而yield()方法不会释放锁。

而我接触到的很多情况是：问线程同步的问题，大多数人基本上只知道synchronized。

要搞清线程的同步问题，大家要先了解一下“对象的同步锁”，这个留给大家自己去看吧，这里不做展开。我们回到新接到的这个需求上来，这个场景其实挺合适做为一个面试题的。

如何实现呢？我想到一个简单的方法就是用CountDownLatch。

CountDownLatch：一个同步辅助类（大名鼎鼎的java.util.concurrent包），在完成一组正在其他线程中执行的操作之前，它允许一个或多个线程一直等待。

用给定的任务数初始化CountDownLatch，一个线程工作完成（任务成功或者失败都算工作完成）就调用 countDown() 方法，当计数到达零之前，await 方法会一直受阻塞。当计数器为零时，会释放所有等待的线程，await后的代码将被执行。

CountDownLatch计数无法被重置。如果需要重置计数，请考虑使用 CyclicBarrier。

还有其他的实现方式吗？这个是肯定的。比如，真接上Thread.jion，代码难看是会难看点，但也能完成这个需求。

我还查到一种方式是使用java.util.concurrent.ExecutorService的awaitTermination阻塞主线程，等待线程池的所有线程执行完成。需要设置一个超时时间的参数，如果超时则awaitTermination返回false，如果线程池中的线程全部执行完成，返回true。

小结

因为现在有很多开源的框架或者代码库，帮我们解决了很多底层诸如网络请求、线程池管理的问题，使得很多情况下我们都不怎么接触到线程同步的问题。不过还有很有必要抽时间来学习一些线程同步的知识，对我们提高并发编程的能力很有帮助。

如果大家有更好的方式实现我开头提到的需求，可以回帖一起讨论一下。

作者：goeasyway

链接：https://www.jianshu.com/p/fd70d652f9e3

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 如何对SQLite数据库中进行大量的数据插入？

Android系统内置了SQLite数据库，并且提供了一整套的API用于对数据库进行增删改查操作。SQLite是一个轻量的、跨平台的、开源的数据库引擎。SQLite每个数据库都是以单个文件（.db）的形式存在，这些数据都是以B-Tree的数据结构形式存储在磁盘上。

使用SQLiteDatabase的insert，delete等方法或者execSQL方法默认都开启了事务，如果操作的顺利完成才会更新.db数据库。事务的实现是依赖于名为rollback journal文件，借助这个临时文件来完成原子操作和回滚功能。

大家可以在/data/data/<packageName>/databases/目录下看到一个和数据库同名的.db-journal文件。

SQLite想要执行操作，需要将程序中的SQL语句编译成对应的SQLiteStatement，比如" select \* from table1 "，每执行一次都需要将这个String类型的SQL语句转换成SQLiteStatement。如下insert的操作最终都是将ContentValues转成SQLiteStatementi：

public long insertWithOnConflict(String table, String nullColumnHack,

ContentValues initialValues, int conflictAlgorithm) {

// 省略部份代码

SQLiteStatement statement = new SQLiteStatement(this, sql.toString(), bindArgs);

try {

return statement.executeInsert();

} finally {

statement.close();

}

} finally {

releaseReference();

}

}

对于批量处理插入或者更新的操作，我们可以重用SQLiteStatement，使用SQLiteDatabase的beginTransaction()方法开启一个事务，样例如下：

try

{

sqLiteDatabase.beginTransaction();

SQLiteStatement stat = sqLiteDatabase.compileStatement(insertSQL);

// 插入10000次

for (int i = 0; i < 10000; i++)

{

stat.bindLong(1, 123456);

stat.bindString(2, "test");

stat.executeInsert();

}

sqLiteDatabase.setTransactionSuccessful();

}

catch (SQLException e)

{

e.printStackTrace();

}

finally

{

// 结束

sqLiteDatabase.endTransaction();

sqLiteDatabase.close();

}

我在华为Nexus 6P上对常见的几种做法做了一下测试。

直接使用SQL语句进行插入

直接使用SQL语句插入，添加事务

使用ContentValues方式，添加事务

使用SQLiteStatement方式，添加事务

结果如下图：

从数据上看，第四种方式使用SQLiteStatement最快，不过只要添加了事务（或者说只需要一个事务，不是每条插入都使用事务），后三种方式的差别并不大。所以针过这个题目的插入的优化可以通过“SQLiteStatement+事务”的方式显著提高效率。

查询方面的优化一般可以通过建立索引。建立索引会对插入和更新的操作性能产生影响，使用索引需要考虑实际情况进行利弊权衡，对于查询操作量级较大，业务对要求查询要求较高的，还是推荐使用索引。所以这会有一个取舍问题，看你的项目是查询频繁还是插入和修改频繁。当然还有一些小的优化细节，如果面试官问到也可以说几点（如limit）。

线程问题

SQLite的同步锁精确到数据库级，粒度比较大，不像别的数据库有表锁，行锁。同一个时间只允许一个连接进行写入操作。

如果有大量的数据处理，那么肯定不合适于在UI线程去操作，这时就要考虑多线程的问题了。我们如果开一个工作线程去操作SQLite数据库，如批量地插入可能需要30秒钟，而这个时间UI线程也要从数据库读取一下数据展示给用户，那么这个时候UI线程能读取到这个数据库吗？大家可以思考一下这个问题。

我们常常在多线程中只使用一个SQLiteDatabase引用，在用SQLiteDataBase.close()的时需要注意调是否还有别的线程在使用这个实例。如果一个线程操作完成后就直接close了，别一个正在使用这个数据库的线程就会异常。所以有些人会直接把SQLiteDatabase的实例放在Application中，让它们的生命周期一致。也有的做法是写一个计数器，当计数器为0时才真正关闭数据库。

使用ORM的问题

目前网上有很多开源的ORM（对象关系数据映射）框架，如greenDAO、ormlite等等。在使用这些框架有必要很了解一下它们的利弊，特别是一些使用反射的框架，对性能的影响会比较大。有些框架在多线程同步方面也会产生一些问题，所以使用时要有所顾虑。

Realm 是最近兴起的一个专注于移动设备数据库的库，其核心是使用C++编写，号称很多时候数据的存取速度比SQLite要快很多。不过在我的一些项目中，发现它读取并不比SQLite快。

小结

在实践中我们总结出一条守则：“不要用Helloworld来测试自己的框架（或代码），要测就要用真实的数据和环境。”

特别是针对数据库方面，如果只用几条简单的数据进行测式，那么你会很容易傲娇和满足，而忽视了很多问题。没有经过真实数据（或大量数据）测试之前，不要对自己的代码太过自信。

作者：goeasyway

链接：https://www.jianshu.com/p/2398aad3bd61

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## Activity的启动模式（launchMode）有哪些，有什么区别？

这应该是一道很虐人的面试题，很多人都答不上来，很多人根本就没有用过。当我发现在被我面试的人中有80%的比例对它不了解时，我找过一些同事讨论是否还有在面试中考查这个问题的必要，得到的回答是“程序员何苦为难程序员”！

因为很多程序员都认为这个启动模式没有多大用处。好吧，我用一个实际中很容易遇到的问题来引出它有多么有用。

很多人在使用startActivityForResult启动一个Activity时，会发现还没有开始界面跳转本身的onActivityResult马上就被执行了，这是为什么呢？

遇到过吧，我见过很多人为了这个问题抓耳挠腮的。在Activity.java的startActivityForResult方法上看一下官方的说明吧：

\* <p>Note that this method should only be used with Intent protocols

\* that are defined to return a result. In other protocols (such as

\* {@link Intent#ACTION\_MAIN} or {@link Intent#ACTION\_VIEW}), you may

\* not get the result when you expect. For example, if the activity you

\* are launching uses the singleTask launch mode, it will not run in your

\* task and thus you will immediately receive a cancel result.

很多人出现这个问题，确实是因为startActivityForResult启动的Activity设置了singleTask的启动模式。但是，除了这种情况还有可能会马上执行吗？

有，而且很多。如下面表格，左边第1列代表MainActivity的启动模式，第一行代表SecondActivity（即要startActivityForResult启动的Activity）的启动模式，打叉代表在这种组合下onActivityResult会被马上调用。

|stand|singleTop| singleTask | singleInstance

----|:------:|:----:|:----:|:----:

stand|√|√|x| x

singleTop|√|√| x|x

singleTask|√|√| x |x

singleInstance|x|x|x|x

好在幸运的是，Android在5.0及以后的版本修改了这个限制。也就是说上面x的地方全部变成了√。

那么在Android 5.0后，还会有这个问题吗？

还是会的。如在Intent中设置了FLAG\_ACTIVITY\_NEW\_TASK再startActivityForResult，即使是标准的启动模式仍然会有这个问题。

intent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

Log如下：

07-12 14:21:14.849 20774-20774/net.goeasyway.test I/MainActivity: onCreate

07-12 14:21:14.875 20774-20774/net.goeasyway.test I/MainActivity: onResume

07-12 14:21:19.995 20774-20774/net.goeasyway.test I/MainActivity: onPause

07-12 14:21:19.995 20774-20774/net.goeasyway.test I/MainActivity: onActivityResult requestCode=1 resultCode=0

07-12 14:21:19.996 20774-20774/net.goeasyway.test I/MainActivity: onResume

07-12 14:21:19.996 20774-20774/net.goeasyway.test I/MainActivity: onPause

07-12 14:21:20.005 20774-20774/net.goeasyway.test I/SecondActivity: onCreate

07-12 14:21:20.018 20774-20774/net.goeasyway.test I/SecondActivity: onResume

注意：MainActivity的onResume也会被触发。因为onActivityResult被执行时，它会重新获得焦点。很多人也会遇到onResume被无故调用，也许就是这种情况。

所以，最终我们发现只要是不和原来的Activity在同一个Task就会产生这种立即执行onActivityResult的情况，从原代码也可以得到验证，详情查看ActivityStackSupervisor.java。

if (r.resultTo != null && (launchFlags & Intent.FLAG\_ACTIVITY\_NEW\_TASK) != 0

&& r.resultTo.task.stack != null) {

// For whatever reason this activity is being launched into a new

// task... yet the caller has requested a result back. Well, that

// is pretty messed up, so instead immediately send back a cancel

// and let the new task continue launched as normal without a

// dependency on its originator.

Slog.w(TAG, "Activity is launching as a new task, so cancelling activity result.");

r.resultTo.task.stack.sendActivityResultLocked(-1,

r.resultTo, r.resultWho, r.requestCode,

Activity.RESULT\_CANCELED, null);

r.resultTo = null;

}

原因

其实上面代码中的英文注解也说得很清楚了，Android认为不同的Task之间对这种要求返回结果的启动方式会产生一些依赖（对Task），所以干脆简单粗暴在跳转前直接返回RESULT\_CANCELED结果。

我们还是用一个例子简单解释一下，如下图，有两个任务栈（stack），处于前可视状态的是“Back Stack”也叫返回栈，处理后台的是“Background Task”。

当“Activity 2”通过startActivityForResult启动“Activity Y”时，“Background Task”中的Activity会被压入返回栈的栈顶。这种情况下，如果没有在跳转前直接返回RESULT\_CANCELED给“Activity 2”，那么按Back键，应该要跳转到“Activity X”，而按Back键“Activity Y”就会调用finish会发送Result给启动它的“Activity 2”。这时就很难搞清楚，到底是“Activity 2”还是“Activity X”应该获得焦点了，会产生一些混乱或是违反的原有的一些约定。

小结

关于启动模式的问题，其实我开始写这个系统的文章时就想介绍它的，不过发现它的水实现太深了，需要用比较长的篇幅才能说明清楚。今天也只是通过一个实际中容易碰到的问题引起大家的关注，也同时引出了“任务”和“返回栈”。

所以，就让程序员多为难程序员一次，进一步的说明请听下回分解。

作者：goeasyway

链接：https://www.jianshu.com/p/cb5c4e5598ed

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## Android资源目录的读取顺序？

Android资源文件可以定义在不同分辨率、屏幕方向、语言等（甚至还有夜间模式），当我们的应用需要使用一个资源，这个资源（图片、Layout或者别的）可能在很多res下的子目录中都存在，那么Android系统是如何确认使用哪一个资源呢？

可以先看一下官方文档提供资源，了解清楚资源目录的配置和命名规则。

这就是一个Android应用的资源查找的顺序问题，其实简单说，在查找时会先去掉有冲突的资源目录（上图第1步），然后再按MCC、MNC、语言等指定的优先级进行查找，直到确认一个匹配资源。根据屏幕尺寸限定符选择资源时，如果没有更好的匹配资源，则系统将使用专为小于当前屏幕的屏幕而设计的资源。

图片放错目录会产生的问题吗？

这一点可能很多人都不会注意，觉得只要往一个drawable目录中放了需要的资源就好了。而我们可以自己做一个简单的测试，把同一个图片资源放在不同的dpi目录，会发现它们使用的内存是不一样的。简单说就是高密度（density）的系统去使用低密度目录下的图片资源时，会将图片长宽自动放大以去适应高密度的精度，当然图片占用的内存会更大。

所以如果能提各种dpi的对应资源那是最好，可以达到较好内存使用效果。如果提供的图片资源有限，那么图片资源应该尽量放在高密度文件夹下，这样可以节省图片的内存开支。

mipmap

在使用Android Studio（应该是从1.1版本开始）创建Android应用项目时，常常会看到系统把ic\_launcher.png图标放在了mipmap-xxhdpi目录下了。那么这个mipmap是什么意思呢？和drawable的对应dpi目录有什么区别呢？

我们知道，drawable文件夹是存放一些xml(如selector)和图片，Android会根据设备的屏幕密度(density)自动去对应的drawable文件夹匹配资源文件。

那么mipmap这个目录有什么用呢？

MIP来源于拉丁文中的multum in parvo，意为在一个小空间里的多数。MIP map(有时候拼写成mipmap)是一种电脑图形图像技术，用于在三维图像的二维代替物中达到立体感效应。

Android对放在mipmap目录的图标会忽略屏幕密度，会去尽量匹配大一点的，然后系统自动对图片进行缩放，从而优化显示和节省资源（使用上面说的mipmap技术）。就目前的版本来说，mipmap也没有完全取代drawable的意思，为了更好的显示效果，官方建议如下类型的图片资源可以放到mipmap目录。

Launcher icons.

Action bar and tab icons.

Notification icons

drawable-nodpi文件夹

这个文件夹是一个密度无关的文件夹，放在这里的图片系统就不会对它进行自动缩放，原图片是多大就会实际展示多大。但是要注意一个加载的顺序，drawable-nodpi文件夹是在匹配密度文件夹和更高密度文件夹都找不到的情况下才会去这里查找图片的，因此放在drawable-nodpi文件夹里的图片通常情况下不建议再放到别的文件夹里面。

res/raw和assets的区别

这两个目录下的文件都会被打包进APK，并且不经过任何的压缩处理。

assets与res/raw不同点在于，assets支持任意深度的子目录，这些文件不会生成任何资源ID，只能使用AssetManager按相对的路径读取文件。如需访问原始文件名和文件层次结构，则可以考虑将某些资源保存在assets目录下。

记得之前的版本（Android 2.2）对放在这两个目录的文件还有大小的限制，1M这样吧，之后的版本没有这个限制了。

和美工的关系

很多时候，UI设计师并不太了解Android的DPI，也不太清楚每种DPI对应的ICON规格。这时Android的开发就需要告诉他们一些规范，并结合自己的在真机上的测试经验给予符合项目的ICON尺寸。对于每种密度下的ICON应该设计成什么尺寸其实Android也是给出了最佳建议，ICON的尺寸最好不要随意设计，因为过低的分辨率会造成图标模糊，而过高的分辨率只会徒增APK大小。

�密度 建议尺寸

mipmap-mdpi 48 \* 48

mipmap-hdpi 72 \* 72

mipmap-xhdpi 96 \* 96

mipmap-xxhdpi 144 \* 144

mipmap-xxxhdpi 192 \* 192

小结

我在面试是常把面试者分做两类。一类是打下手的开发，做一些功能或业务模块实现，修修BUG保证应用正常运行；另一类是主创型开发，也可以叫主力开发，他们对应用开发的各个模块都有了解，可以熟练地从零开始搭建出整个项目框架。

而现在面试者一但有几年工作经历，往往在简历和交谈中夸大他们对项目的贡献，本来只是一个打下手的却被抬高成了主力开发。一个人打过多年下手，也会积累很多开发经验，但并不等于他就能适应主力开发的工作，主力选手除了经验的积累外更重要的是思维方式的改变，注重接口的设计胜过功能代码的实现。

如果我们要招一个有经验的主力型选手，还是很有必要通过一个细节来检验他们的。不过还是要提醒一下面试官，对细节的提问要注意到必需的知识和冷门少用到的知识，问对方过于偏僻的细节点往往起不到效果。

作者：goeasyway

链接：https://www.jianshu.com/p/46ce37b8553c

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 有没有遇到Bitmap OOM的问题？如何优化图片占用的内存空间？

我觉得在回答上述问前，我们首先要回答一个问题：

Android系统上的图片指的是什么？

首先看Android支持的图片格式：

Android支持JPEG和PNG格式的图片，4.0之后添加了WEBP图片的支持（不过有些手机产商并不一定支持，如小米2S 4.1的SDK版本，但还是显示不了WEBP格式），5.0之后增加了对GIF和BMP图片的支持。

其实图片最终要显示在屏幕上，都会对应一个屏幕上的点，即对应一个颜色值。不同格式的图片，只是不同压缩编码和解压算法。也就是说，我们看到的.jpg、.png图片的文件大小只有几十KB，担把它们加载到内存中时，每张图片最终都按长X宽展开，计算其占用内存大小的就变成了（ARGB\_8888格式的图片，每像素占用 4 Byte，而 RGB565则是 2 Byte，假设是ARGB\_8888）：

内存占用=长 X 宽 X 4bytes

这种算法其这还忽略屏幕的Density，如我们上Day 21的面试题中也提到过的，放在不同的drawable目录中的图片显示时会根据Denisty有一定的缩放。所以有时候图片占用的内存会比我们上面公式计算出来的还要大很多。

Bitmap和Drawable

Bitmap是Android系统中的图像处理的最重要类。可以简单地说，Bitmap代表的是图片资源在内存中的数据结构，如它的像素数据，长宽等属性都存放在Bitmap对象中。Bitmap类的构造函数是私有的，只能是通过JNI实例化，系统提供BitmapFactory工厂类给我们从从File、Stream和byte[]创建Bitmap的方式。

Drawable官文文档说明为可绘制物件的一般抽象。View也是可以绘制的，但Drawable与View不同，Drawable不接受事件，无法与用户进行交互。我们知道很多UI控件都提供设置Drawable的接口，如ImageView可以通过setImageDrawable(Drawable drawable)设置它的显示，这个drawable可以是来自Bitmap的BitmapDrawable，也可以是其他的如ShapeDrawable。

也就是Drawable是一种抽像，最终实现的方式可以是绘制Bitmap的数据或者图形、Color数据等。理解了这些，你很容易明白为什么我们有时候需要进行两者之间的转换。

优化手段

在Android 3.0之前的版本，Bitmap像素数据存放在Native内存中，而且Nativie内存的释放是不确定的，容易内存溢出而Crash，所以一般我们不使用的图片要调用recycle()。

从3.0开始，Bitmap像素数据和Bitmap对象一起存放在Dalvik堆内存中（中从源代码上看是多了一个byte[] buffer用来存放数据），也就是我们常说的Java Heap内存。

除了这点改变之外，3.0版本的还增加了一个inBitmap属性（BitmapFactory.Options.inBitmap）。如果设置了这个属性则会重用这个Bitmap的内存从而提升性能。但是这个重用是有条件的，在Android4.4之前只能重用相同大小的Bitmap，Android4.4+则只要比重用Bitmap小即可。

当然优化的手段还有很多，比如使用采样率（inSampleSize），如果最终要压缩图片，如显示缩列图，我们并不需要加载完整的图片数据，只需要按一定的比例加载即可；使用Matrix变形等，比如使用Matrix进行放大，虽然图像大了，但并没有占用更多的内存。

使用第三方图片库也是一种优化吧，它们帮我们完成了很多工作。Facebook的Fresco还自己开辟了Native内存用于存储图片，以得到更大的内存空间（其实我还不确认这种方式是否有兼容性问题）。

要加载很大的图片怎么办？

如果图片很大，比如他们的占用内存算下来就直接OOM了，那么我们肯定不能直接加载它。解决主法还是有很多的，系统也给我们提供了一个类BitmapRegionDecoder，可以用来分块加载图片。

小结

虽然网上有很多开源的和图片相关的库，但我们还是有必要了解一下Bitmap的相关的知识的，有时候我们只需要一个简单的功能，可以手动自己实现，不一定非要加进一个开源库。代码越是简单，可控性和可维护性越好，对吧。

作者：goeasyway

链接：https://www.jianshu.com/p/3c597baa39e5

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## Android中Java和JavaScript如何交互？

先说一些基础的知识，而JavaScript 是一种脚本语言，个人认为他比Java更面像对象，它没有编译、链接等操作，在运行时才动态的进行词法、语法分析，生成抽象语法树和字节码，然后由解释器负责执行或者使用 JIT 将字节码转化为机器码再执行。整个流程由 JavaScript 引擎负责完成，在Android手机上，这个JavaScript 引擎就是WebView的实现内核。

在Android 4.4版本中，原本基于Android WebKit的WebView实现被换成基于Chromium的WebView实现。

在Android应用中我们可以通过WebView从url加载网页，很多公司在面试时也会考查一下面试者是否有Java和JavaScript交互的经验。这里我们也简单说一下，Java和JavaScript交互主要分为：

1、Java调用WebView加载的网页上的JavaScript

在Java端可以通过WebView调用JavaScript，基本格式为：

webView.loadUrl(“javascript:methodName(parameterValues)”)

有时还要注意这个javascript:methodName是有参数还是无参数，有返回值的情况也会有所不同，毕竟我们从webView.loadUrl的接口看不到如何处理返回值。Android 4.4之后WebView增加了evaluateJavascript接口可以供我们通过ValueCallback这个回调处理返回值。

public void evaluateJavascript(String script, ValueCallback<String> resultCallback)

在4.4之前的版本，常用的思路是Java调用JavaScript方法，JavaScript方法执行完毕，再次调用Java代码将值返回。

2、JavaScript调用本地的Java对像方法

JavaScript调用Java需要在WebView通过addJavascriptInterface添加一个实例到WebView的一个Map对象中，也就是下面接口中第一个参数。

public void addJavascriptInterface(Object object, String name)

调用格式为window.�name.methodName(parameterValues)，methodName为object实例提供的方法。在Android 4.2之后引入了@JavascriptInterface，被标为JavascriptInterface的方法才能在JavaScript端进行调用，也就是说在4.2之前的版本开放JavaScript调用Java实例的功能是有全安隐患的。

因为JavaScript端调用Java时是通过反射的方式。可以在JavaScript端通过window.�name.getClass().forName反射调用java.lang.Runtime类的方法（getRuntime），之后便可以执行一些命令做些坏事情了。

所以在4.2之前的版本一般都建议：

关闭JavaScript功能（setJavaScriptEnabled）；

不使用addJavascriptInterface；

用removeJavascriptInterface移除系统自带的实例（如：WebView默认会addJavascriptInterface添加一个 “searchBoxJavaBridge\_” 的实例）。

如果非要在4.2之前的版本使用JavaScript功能，那该怎么办呢？这个问题我也没有很好的答案，你可以试试问问面试者，看看他们的想法。

混合开发（Hybrid App）的问题

人都是向往懒和舒服的，当我们针对同一业务需求要开发多个版本的移动端应用时，我们很容易想到能不能像JAVA一样只写一次代码在各个平台都可以运行，所以就产生了一种叫Hybrid App（混合应用）开发方式。

简单地说，原来一个项目我们一般要做Web、Android和IOS三个应用，现在我们做混合应用，用CSS，HTML和JavaScript编写应用，在Android和IOS上用Webview做为载体来运行。很多开源库也为我们写为了JavaScript调用原生模块的接口供我们调用。只要我们的HTML界面看起来像手机自带的控件，那么理论上用户是区分不开是原生的应用还是Web应用，而且可以动态部署的优点也很吸引人。

但理想是丰满的，现实总是那么残酷。

性能问题首当其冲，IOS的手机要好些，但在硬件标准参差不齐的Android设备上，使用Webview加载和渲染JavaScript的性能较差，稍微复杂一点或CSS动画多一些，卡顿现象就很常见了。

而且兼容性问题也很突出，包括Android官方的WebView实现内核都发生了变化，每个版本所支持的功能也有不同。再加上ROM厂商乱改系统自带的WebView，从而导致在各种小细节上不同手机的显示效果或运算结果不同。

安全问题和调试难的问题也不容忽视，最终发现解决它们成本并不比原生开发的成本低。之前Facebook也想使用H5来做移动应用，后来也无法解决这些问题，最终放弃了，所以才有了今天的React Native。

React Native

正如项目的名字那样，React Native的目的是构建真正native的应用。而不是构建在Webview里运行的混合模式的应用，开发完全由JavaScript和React来完成。简单说它的原理就是，开发用JavaScript开发Web的方式进行开发，最终在移动端会使用一个JavascriptCore解释器引擎来解析JS相关文件成相应的原生控件再进行渲染，性能上得到了很大的提升。

FaceBook开发React是对以前思维模式进行了改变，不再追求一次编写到处运行，而是转而探讨：

Learn once，Write anywhere!

小结

前端的发展如火如荼，而且从业人数不可小觑，目前高端的前端开发的薪资待遇方面都很不错，也说明了市场对它的认可。虽然目前看，前端开发还有些混乱（开发框架和模式琳琅满目），难度和复杂度也在增加，但并不能排除它一统天下的可能性。

做为Android的工程师，你无法确认Androd或者IOS会不会走Symbian（Nokia用的平台）的老路，这样看来，接触一些前端开发知识，对我们来说是很有必要的。这也是为什么我会因为一个高级工程师不知道React Native而改变对他的看法。

作者：goeasyway

链接：https://www.jianshu.com/p/4bb5672ff88d

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 两个Fragment之间如何进行通信？

先说一下Fragment，中文可以叫片段，其实我们也可以把它理解成一个组件，它有自己的生命周期（回调函数），可以组织UI和业务逻辑。但它不像Android的四大组件（Activity, Service, BroadcastReceiver和ContentProvider）那样可以独立存在，它必须要依附于Activity，由Activity中的FragmentManager实例来管理它的生命周期。

Google最初引入Fragment主要是为了给大屏幕（如平板电脑）上更加动态和灵活的 UI 设计提供支持，看一下下图你就明白了：

我们可以在多个Activity中重复使用某个片段，并且可以按我们的需求把不同的片段组合在一个界面中，好处显而易见，减少工作量提高了代码的可重用性。

也就是说每个片段是相互独立的，如上图所示，Fragment A并不知道这个Activity中是否存在Fragment B。这样一来就会产生Fragment之间通信的问题（比如Fragment A点击了列表的一项，Fragment B的内容要进行更新），当然这个问题的答案并不是唯一的，有很多种做法。不过，最基础的做法你应该了解一下，就是通过Activity进行通信。

尽管Fragment是作为独立于Activity的对象实现，并且可在多个Activity内使用，但片段的给定实例会直接绑定到包含它的 Activity。具体地说，片段可以通过getActivity()访问Activity实例。

同样地，Activity也可以使用findFragmentById()或findFragmentByTag()，通过从FragmentManager获取对Fragment的引用来调用片段中的方法。

知道这些基础，你应该很清楚怎么进行通信了。不过我们建议的方式是面向接口编程，在Fragment中getActivity获取到的Activity实例应该是实现了我们某个接口的实例，即在Fragment的代码中不应该出现某个具体的Activity类。

Fragment的生命周期

如果你喜欢使用Fragment，一定要清楚这些生命周期的方法，哪个方法会创建视图（View），哪个会销毁视图，这样才能更好的使用它们。

先来看看官方的Fragmenet生命周期和Activity生命周期的对应关系图：

Activity 生命周期对片段生命周期的影响

是不是觉得，好像也没有多复杂，只是在一些对应的Activity生命周期方法上进行了细分。

我们再看看反对使用Fragment的人是如何看待Fragment的生命周期的：

是不是头都大了？简单地说Fragment的生命周期并不像官方描述的那样简单，感兴趣的朋友可以看看这篇文章：我为什么主张反对使用Android Fragment。

Fragment虽然在某种程度上实现了代码复用，但并没有将UI和逻辑分离，而且引入了复杂的生命周期，在一定程度上也限制了它的使用。如Fragment嵌套Fragment可能会遇到嵌套的Fragment接收不到onActivityResult事件之类的问题。

不过，我也看到有人是这样使用Fragment的，创建一个没有UI的Fragment仅将它做为保存状态的工具使用。因为Fragment也有onSaveInstanceState方法，可以在里面对状态进行处理，而且系统回收Activity后再使用Activity时会帮我们恢复Fragment的状态。

关于Fragment还有很多细节，如果面试者经常使用的话，还可以问问“如何管理Fragment回退栈”或者“FragmentTransaction中remove和detach的区别”等问题。

小结

其实我也不喜欢用Fragment，因为我发现写Fragment和写Activity的感觉是一样的，UI加逻辑，也许是在为我面对的很多项目都不需要考虑平板的原因。不过即使考虑平板，我们一样可以通过自定义的View去实现。有时候我们使用Fragment，是因为考虑到使用它进行界面切换比Activity切换的代价要小很多。

面试时，面试者不一定能记起所有的细节，但我认为一些基础机制（或原理或者默认习俗）都是应该了解的，不然我会认为面试者不是经历的项目太少，就是没有进行系统性的学习，简单说就是，你可以没有使用过Fragment（这并不丢人），但你一定要了解一下Fragment是什么东西吧？

作者：goeasyway

链接：https://www.jianshu.com/p/c54db57a5bd4

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 如何理解Android应用的进程？

进程是一个动态的过程，每一个App的运行都是在一个独立的进程中，进程有自己独立的内存和数据空间，进程的名字就是App的packageName，这些进程都是从Zygote进程Fork出来的，并受AMS（ActivityManagerService）管理。

默认情况下，同一App的所有组件均在相同的进程中运行，但也可以允许App有多个进程。在AndroidManifest.xml里边给四大组件配置android:process属性，就可以让这些组件在指定的进程中运行，这些进程名字都是packageName:name这种，以区分是属于哪个App，我一般称之为辅助进程。

此外，您还可以设置android:process（即不以“:”开头，以符合包命名的格式字符串为它的值），使不同应用的组件在相同的进程中运行，但前提是这些应用共享相同的Linux用户ID并使用相同的证书进行签署。

常有些开发不知道为什么自己的Application.onCreate中的代码执行了两次，如果你遇到这样的情况可以检查一下AndroidManifest.xml是否给某个组件配置了android:process属性。每个进程创建后，都会启动一个主线程（Looper接收消息），每个组件启动前都会先创建Application实例（一个进程只创建一个）。

优先级

当系统内存不足时，Android系统会选择终止掉一部份进程，回收其所占用的内存空间。 为了确定保留或终止哪些进程，系统会根据进程中正在运行的组件以及这些组件的状态，将每个进程放入“重要性层次结构”中。 必要时，系统会首先消除重要性最低的进程，然后是重要性略逊的进程，依此类推，以回收系统资源。

重要性从高到低如下所示：

前台进程

可见进程

服务进程

后台进程

空进程

其实也可以用优先级（oom\_adj值）来表示这个重要性。App进程的优先级是在com.android.server.am.ProcessList

类里边定义的。所以当内存不足的时候，进程优先级低的（oom\_adj越大的，系统进程是负数），占内存大的App进程将会被优先kill掉。

很多项目都比较“变态”，总想长期霸占内存（如消息推送），所以就有了各种进程保活的黑科技面市。黑科技我们占且不论，对于一般提高进程优先级的方法，大家还是应该了解一些。

如：

进程要运行一些组件，不要成为空进程。

远行一个Service，并设置为前台运行方式（startForeground）。

AndroidManifest.xml中配置persistent属性（persistent的App会被优先照顾，进程优先级设置为PERSISTENT\_PROC\_ADJ=-12）

关于第2点，摘抄一段代码给大家看：

private void keepAlive() {

try {

Notification notification = new Notification();

notification.flags |= Notification.FLAG\_NO\_CLEAR;

notification.flags |= Notification.FLAG\_ONGOING\_EVENT;

startForeground(0, notification); // 设置为前台服务避免kill，Android4.3及以上需要设置id为0时通知栏才不显示该通知；

} catch (Throwable e) {

e.printStackTrace();

}

}

在Service的onCreate方法调用keepAlive()即可，其实就是是欺骗系统把自己当成一个一直在通知栏的Notification。不过这种方式，并不保证在所有的机型上都有效。

线程

线程是CPU调度的基本单元，一个应用都有一个主线程负责处理消息。一个应用启动后，至少会有3个线程，一个主线程（UI线程）和2个Binder线程。Zygote进程（APK所在的进程也是由Zygote进程Fork出来的）还会产生有一些Daemon线程如：ReferenceQueueDaemon、FinalizerDaemon、FinalizerWatchdogDaemon、HeapTaskDaemon，从名字大家也可以对它们的用途猜出一二。

线程间可以共享资源，为了保存UI的更新不会混乱，所以更新UI控件时要求在主线程进行更新，即需要保证更新UI是线程安全的。有时还可以问问面试者，什么是线程安全，貌似不知道这个概念的人也不少。

线程同步方面，之前章节有介绍过，这里就不在讲了。

小结

关于进程其实有很多可以讲的，比如DVM, Native进程，或者讲讲Linux进程方面的知识也可以。面试的时候，并不一定需要面试者回答标准答案，或者一定要答对。很多时候，是要面试者对问题进行思考并用自己的语言把自己的理解表达出来，对错并不是特别重要。

作者：goeasyway

链接：https://www.jianshu.com/p/999650fda571

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 如何解决ScrollView嵌套中一个ListView的滑动冲突？

后来我一试，发现ScrollView布局中嵌套Listview显示是不正常的，确切地说是只会显示ListView的第一个项。

先说下为什么会只显示ListView的第一个Item，简单的说就是ListView在计算（比较正式的说法是：测量）自己的高度时对MeasureSpec.UNSPECIFIED这个模式在测量时只会返回一个List Item的高度（当然还有一些padding这些的值我们可以先忽略），而ScrollView的重写了measureChildWithMargins方法导致它的子View的高度被强制设置成了MeasureSpec.UNSPECIFIED模式。

ListView.java的onMeasure()代码片段：

if (heightMode == MeasureSpec.UNSPECIFIED) {

heightSize = mListPadding.top + mListPadding.bottom + childHeight +

getVerticalFadingEdgeLength() \* 2;

}

ScrollView.java的measureChildWithMargins()代码片段：

final int childHeightMeasureSpec = MeasureSpec.makeMeasureSpec(

lp.topMargin + lp.bottomMargin, MeasureSpec.UNSPECIFIED);

child.measure(childWidthMeasureSpec, childHeightMeasureSpec);

注意：ScrollView继承于FrameLayout，但它的布局中只能有一个子View，常用的是LinearLayout。

说到这里，我们肯定要来看看MeasureSpec是什么东西，而且这也是一个很好的面试题，如果做过自定义View，对它肯定不会陌生的。我们在XML在布局文件中，设置布局的高和宽时，常常会用到“100dp”、“wrap\_content”或者“match\_parent”这类的值去设置它的android:layout\_width和android:layout\_height，而对于每个View控件来说，这两个值都是必需的。

最终我们把View绘制到屏幕时，需要将View的宽高值映射到屏幕上的像素大小，这就要在draw前先确定本身的宽高和每个子布局的具体宽高（像素值），这中间就需要一个转换的过程，如把wrap\_content转换成100px，这就是measure的工作。

而布局中有很多子布局，或者说ViewGroup中可能会有多个ViewGroup和View，整个测量过程也是一次根结点开始的遍历过程，在这个过程中父布局需要告诉它的子布局具体的模式和宽高值（对子布局是一种约束，子布局需要在允许的范围内绘制），最终Android用一个int型来表示模式和值。

做过手机游戏的一定很容易想到用位移。int占4个字节，32位（bit），前2位（高位）用于存Mode，后面30位用于存宽高的具体值。当然了我们不用具体去操作，有一个封装好的MeasureSpec类会帮我们处理这些事情。这就是为什么我们看别人的自定义UI源码时常常看到如下的代码：

@Override

protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

int widthMode = MeasureSpec.getMode(widthMeasureSpec);

int heightMode = MeasureSpec.getMode(heightMeasureSpec);

int widthSize = MeasureSpec.getSize(widthMeasureSpec);

int heightSize = MeasureSpec.getSize(heightMeasureSpec);

Size为具体的值，而Mode就是我们说的三种模式：UNSPECIFIED，EXACTLY和AT\_MOST。

UNSPECIFIED

不限定，父View不限制子View的具体的大小，所以子View可以按自己需求设置宽高（前面说的ScrollView就给子View设置了这个模式，ListView就会自己确认自己高度）。

EXACTLY

父View决定子View的确切大小，子View被限定在给定的边界里，忽略本身想要的大小。

AT\_MOST

最多的，子View最大可以达到的指定大小（当设置为wrap\_content时，模式为AT\_MOST, 表示子view的大小最多是多少。）

知道了这些我们解决这个问题，就不算难了，我们也可以重写ListView的onMeasure让它按我们的要求测量高度。

显示正常之后，遇到了91面试官和我说的滑动事件冲突问题，ScrollView和ListView都是上下滑动的，嵌套在一起后ScrollView中的ListView就没法上下滑动了，事件被ScrollView响应了。

就里又引出了一个常被问到的面试题：ViewGroup的Touch事件分发机制。我们触摸幕时会产生事件（MotionEvent）：

ACTION\_DOWN：手指开始触摸到屏幕的那一刻响应的是DOWN事件；

ACTION\_MOVE：接着手指在屏幕上移动响应的是MOVE事件；

ACTION\_UP：手指从屏幕上松开的那一刻响应的是UP事件。

事件的分发中我们较关注的三个方法：

分发事件：dispatchTouchEvent

在这里进行事件的分发，onInterceptTouchEvent和onTouchEvent都是由dispatchTouchEvent负责调度的。

拦截事件：onInterceptTouchEvent

只有ViewGroup才有这个方法。拦截了的话，ViewGroup就不会把事件继续分发给子View了，即子View的dispatchTouchEvent和onTouchEvent这两个方法都不会被调用。返回true时，表示ViewGroup会拦截事件。

消费事件：onTouchEvent

onTouchEvent 返回true时，表示事件被消费掉了。一旦事件被消费掉了，其他父元素的onTouchEvent方法都不会被调用。

用一张图简单说明一下分发的的大体流程：

现在我们回过头来看，ScrollView和ListView的事件冲突问题，从ScrollView的源码可以看到它对Touch事件（ACTION\_MOVE）进行了拦截，所以滑动的事件传递不到ListView。

所以我们解决这个问题，需要让在ListView区域的滑动事件ScrollView不要拦截。这样在ListView区域外的还是由ScrollView去处理事件，ListView外滑动的就是ScrollView。这里用到一个系统自带的API来实现这种方案：requestDisallowInterceptTouchEvent（我觉得可以从名字直接读出它的用途，不再解释），代码也不复杂:

public class MyListView extends ListView {

public MyListView(Context context) {

super(context);

}

public MyListView(Context context, AttributeSet attrs) {

super(context, attrs);

}

public MyListView(Context context, AttributeSet attrs, int defStyleAttr) {

super(context, attrs, defStyleAttr);

}

@Override

protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

int newHeightMeasureSpec = MeasureSpec.makeMeasureSpec(480, // 固定高度（实际中这个值应该是根据手机屏幕计算出来的）

MeasureSpec.AT\_MOST);

super.onMeasure(widthMeasureSpec, newHeightMeasureSpec);

}

@Override

public boolean onInterceptTouchEvent(MotionEvent ev) {

switch (ev.getAction()) {

case MotionEvent.ACTION\_DOWN:

case MotionEvent.ACTION\_MOVE:

getParent().requestDisallowInterceptTouchEvent(true);

break;

case MotionEvent.ACTION\_UP:

case MotionEvent.ACTION\_CANCEL:

getParent().requestDisallowInterceptTouchEvent(false);

break;

}

return super.onInterceptTouchEvent(ev);

}

}

小结

关于这部份其实还是有很多可以讲的，但并不一定适合拿来做面试题，我觉得它们太偏细节了，很多地方自己久不做了也不一定说得出来（甚至说错都可能）。而且，这种细节方面的问题可以编写代码时就发现，不容易产生问题，不过对事件的分发机制有一个大体的了解还是很有必要的。

作者：goeasyway

链接：https://www.jianshu.com/p/9abf6a874feb

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 谈谈你使用过的Android开源库，是否有遇到过什么问题？

前段时间和京东的一位Android工程师在聊使用第三方库的问题，他也喜欢问面试者在使用第三方库时有遇到什么问题和解决的方案，他给我举了个关于网络请求库OKHttp的例子（OKHttp这个开源框架处理请求转发时会有问题）。如果对方在网络开发上很有经验，比如像他们做电商移动应用的，是很容易遇到这样的问题。

前几年在一个项目中，我有使用EventBus事件总线框架做为应用UI和逻辑沟通的桥梁，后来也发现并不像我想象中的好用。太多的消息事件会让代码的可读性和可维护性降低，当时EventBus还使用反射的方式实现，性能上也有一定的消耗（EventBus升级3.0以后事件的订阅已经从方法名换成了注解的方式）。

EventBus是一款针对Android优化的发布/订阅事件总线（内部是观察者设计模式）。主要功能是替代Intent,Handler,BroadCast在Fragment，Activity，Service，线程之间传递消息.优点是开销小，代码更优雅。以及将发送者和接收者解耦。

当然，�每个人遇到的问题都不一样，有些别人用着很正常的开源库你用着就可能会出问题。比如我遇到的Glide图片加载库在初始化Glide实例时写死了Context的问题，在一般的应用中这样做并不会出现问题，但在插件框架上就会因为这个Context被写死就会导致插件资源错乱的问题。

Glide.with(this).load("http://goo.gl/gEgYUd").into(imageView);

Glide在初始化时记录了一个Context做为其内部使用，之后虽然每次都会通过“with(this)”传入一个Context，但这个新的Context却不会被使用。

还有一些项目中使用ORM（对象关系映射）的数据库框架，如AFinal、GreenDao，有些框架在多线程操作数据时会锁住整个表（在ORM的概念里，其实就是Dao类），导致在初始化时因为子线程在锁住某个类阻塞了UI线程的数据表操作。

如果没有问题呢？

可不可以在使用第三方开源库时，就是没有问题呢？当然可以，不过你可以聊一聊你为什么要选择这个库，其实就是从侧面来说明：你对这个库的优缺点的看法，以及你是否了解它的实现原理，因为你要对自己的项目负责（开源作者可不需要）。

小结

这里举了几个小例子，还有很多优秀的开源框架。其实很多程序员的问题是很少使用开源库或者框架，除了图片加载库是必备之外，其他的开源库很多人就不一定会使用了，自然也不会遇到什么问题。

为什么我会比较看重这一点呢？因为，我觉得第三方的开源库，你不一定要在自己的项目中使用，但你一定要去学习别人是怎么写这个库的，是不是比你的方法更好，而且很多流行的开源库都是一些Android大牛写的，直接读和修改他们的代码是你接近他们最简单的方式。

作者：goeasyway

链接：https://www.jianshu.com/p/571d9a4e51cf

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## AIDL是什么？你有使用过它吗，它支持哪些数据类型？

AIDL是Android Interface Definition Language的简写，即Android接口定义语言。我们知道Android系统为每一个应用开启一个独立的虚拟机，每个应用都运行在各自进程里（默认情况下），彼此之间相互独立，无法共享内存。当一个应用想要访问另一个应用的数据或调用其方法，就要用到Android系统提供的IPC机制。而AIDL就是Android实现IPC机制的方式之一。

除了AIDL，Android还提供了Messenger来实现跨进程通信，不过Messenger是以单线程串行方式（消息队列）来处理来自不同客户端的访问的，并不适合多线程并发访问。当需要提供跨进程以及多线程并发服务时就需要AIDL上场了。

Messenger实际上也是以AIDL作为其底层结构。

其实要应对AIDL相关的面试题，除了了解清楚它的作用和注意的事项外，最有效的手段莫过于自己动手写几次。这里给大家定两个小目标：

会使用AIDL进行进程间通信；

会手写AIDL的编码，加深对Binder机制的理解。

创建AIDL

我们先实现第一个，在Android Studio中创建一个简单的AIDL项目，实现IPC通信。

Step1. 创建.aidl文件

我们在对应的src的Package下创建一个AIDL文件（Android Studio->File->New->AIDL->AIDL file），创建后Android Studio会自动把这个.aidl文件放到一个aidl的目录下。

Android SDK Tool会根据我们的.aidl文件自动生成一个同名的.java文件，如：AIDLTest/app/build/generated/source/aidl/debug/net/goeasyway/aidltest/IRemoteService.java

basicTypes方法中给我们展示了AIDL支持的基本数据类型，除此之外，AIDL还支持：CharSequence, List & Map（List和Map中的所有元素都必须是AIDL支持的数据类型、其他AIDL生成的接口或您声明的可打包类型。）

Step2. 创建一个Service暴露AIDL接口并实现AIDL的接口函数

如下代码，创建一个Service:

public class RemoteService extends Service {

public RemoteService() {

}

@Override

public IBinder onBind(Intent intent) {

return binder; //暴露给客户端

}

// 实现AIDL接口

private final IRemoteService.Stub binder = new IRemoteService.Stub() {

@Override

public int getPid() throws RemoteException {

return Process.myPid();

}

@Override

public void basicTypes(int anInt, long aLong, boolean aBoolean, float aFloat,

double aDouble, String aString) throws RemoteException {

}

};

}

然后在MainActivity通过bindService绑定这个服务，即可以获得AIDL的接口调用的引用。运行前，我们设置一下AndroidManifest.xml文件记这个Service运行在一个单独的进程中：

<service

android:name=".RemoteService"

android:process=":remote"

android:enabled="true"

android:exported="true"/>

现在来bindService:

public class MainActivity extends AppCompatActivity {

private final static String TAG = "MainActivity";

private IRemoteService remoteService;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Intent intent = new Intent();

intent.setClass(this, RemoteService.class);

bindService(intent, connection, Service.BIND\_AUTO\_CREATE); // 绑定服务

}

private ServiceConnection connection = new ServiceConnection() {

public void onServiceConnected(ComponentName className, IBinder service) {

remoteService = IRemoteService.Stub.asInterface(service); //获取AIDL的接口实现引用

try {

Log.i(TAG, "Client pid= " + Process.myPid());

Log.i(TAG, "RemoteService pid= " + remoteService.getPid());

} catch (RemoteException e) {

e.printStackTrace();

}

}

public void onServiceDisconnected(ComponentName className) {

Log.e(TAG, "Service has unexpectedly disconnected");

remoteService = null;

}

};

}

从输出的日志看到Service和Activity运行在两个不同的进程中：

02-05 09:51:40.154 18992-18992/net.goeasyway.aidltest I/MainActivity: Client pid= 18992

02-05 09:51:40.154 18992-18992/net.goeasyway.aidltest I/MainActivity: RemoteService pid= 19022

到这里，我们完成了一个AIDL的范例，有几个地方可能会对我们造成困扰：

Stub类：Binder的实现类，服务端需要实现这个类来提供服务。

asInterface函数： 一个静态函数，用来将IBinder转换成对应的Binder的引用。先通过queryLocalInterface查询，如果服务端和客户端都是在同一个进程，那么就不需要跨进程了，直接将IRemoteService当做普通的对象来使用，否则会返回远程对象的代理对象（Proxy）。

public static net.goeasyway.aidltest.IRemoteService asInterface(android.os.IBinder obj)

{

if ((obj==null)) {

return null;

}

android.os.IInterface iin = obj.queryLocalInterface(DESCRIPTOR);

if (((iin!=null)&&(iin instanceof net.goeasyway.aidltest.IRemoteService))) {

return ((net.goeasyway.aidltest.IRemoteService)iin);

}

return new net.goeasyway.aidltest.IRemoteService.Stub.Proxy(obj);

}

通过IPC传递对象

现在，我们加大一下难度，AIDL的接口使用一个我们自己定义的类为参数（或者返回值）。实现步骤如下：

添加一个自定义对象类，并且要实现Parcelable接口，如MyProcess.java；

在AIDL目录下的相同Pacage下添加一个同名的AIDL文件，如MyProcess.aidl；

注意：通过“Android Studio->File->New->AIDL->AIDL file”不让你创建和MyProcess.java同名的AIDL文件，你可以直接用通过“Android Studio->File->New->File”创建一个MyProcess.aidl。

在AIDL接口类中添加一个接口函数，使用MyProcess做为参数或者返回值；

其他的细节大家可以直接查看Github上的代码：https://github.com/goeasyway/AIDL\_Test （或者查看提交的说明找到具体每次的代码区别：https://github.com/goeasyway/AIDL\_Test/commits/master ）

in、out & inout

这节我们看到“MyProcess getProcess(in MyProcess clientProcess);”这个接口的参数有一个“in”修饰符，这也是一个常见的面试题，可以考察一下对方是否真的写过AIDL的代码。

问题：AIDL中的接口函数有时会使用in、out或者inout的参数修饰符，它们各表示什么意思？在什么情况下要使用呢？

in、out和inout表示数据的流向。大家可以把AIDL的客户端和服务端理解成两个进程（其实大多数情况也是这样才会使用AIDL），从客户端流向服务端用in表示，表示这个对象是从客户端中传递到服务端，在服务端修改这个对象不会对客户端输入的对象产生影响。

而out则表示，数据只能从服务端影响客户端，即客户端输入这个参数时，服务端并不能获取到客户端的具体实例中的数据，而是生成一个默认数据，但是服务端对这个默认数据的修改会影响到客户端的这个类对象实例发生相应的改变。

理解了in、out之后，inout自然不需要再解释了。AIDL默认支持的数据类型使用in修饰符，对于我们自定义的Parcelable对象，一般情况下我们也是使用in，如果没有必要，应该尽量避免inout。

Intent也是Parcelable实现

也许你会想到，我们在Activity间可以通过Intent携带参数，其实你去看的源码的话会发现Intent也是一个Parcelable的实现类，而且在系统的工程中也有一个Intent.aidl文件（路径：/frameworks/base/core/java/android/content/Intent.aidl）。所以，它才可以在进程间传递。

注：目前Client端和Server端在同一工程中，如果分开在不同的工程的话，Client端所在的工程要把Server端提供的.aidl复制到同名Pacage的AIDL代码目录下。

手动方式创建AIDL（不依赖AIDL工具，手写远程AIDL的代码完成跨进程通信）

通过AIDL，可以让本地调用远程服务的接口就像调用本地接口那么简单，让用户无需关注内部细节，只需要实现自己的业务逻辑接口，内部复杂的参数序列化发送、接收、客户端调用服务端的逻辑，用户并不需要关心。

AIDL的代码生成器，已经根据.aidl文件自动帮我们生成Proxy、Stub（抽象类）两个类，并且把客户端代理mRemote的transact()过程以及服务器端的onTtransact()过程默认实现好了，我们只需要在服务端继承Stub，实现自己的业务方法即可。

但现在，为了进一步加深对Binder机制的理解，我们来做一个手动实现编写AIDL相关代码的练习。

具体的代码大家可以参考：https://github.com/goeasyway/AIDL\_Test/tree/master/app/src/main/java/net/goeasyway/aidltest/diy

这个包里有三个类：IRmote.java为接口，Stub.java为Binder实现类（Service端要实例化它并在onBind返回），Proxy.java为代理类，提供给客户端使用的，通过Binder驱动和服务端通信。

大家可以看到这个练习不需要.aidl文件。

在这几个代码中，大家需要搞清楚这个类（或者接口）的关系：

Binder

Binder本地对象。

IBinder

IBinder是一个接口，它代表了一种跨进程传输的能力。

IInterface

IBinder负责数据传输，那么client与server端的调用契约呢？这里的IInterface代表的就是远程server对象具有什么能力。具体来说，就是aidl里面的接口。

Proxy

代表远程进程的Binder对象的本地代理，继承自IBinder，因而具有跨进程传输的能力。实际上，在跨越进程的时候，Binder驱动会自动完成代理对象和本地对象的转换。

Stub

这个类继承了Binder, 说明它是一个Binder本地对象，它实现了IInterface接口，表明它具有远程Server承诺给Client的能力；Stub是一个抽象类，具体的IInterface的相关实现需要我们手动完成。

如果觉得有点难理解的话，不妨先动手写了再来看。

小结

如何使用AIDL应该是一个高级工程师必备技能，如果你之前不太了解它的话，那么我强烈建议你完成上面的两个小练习，再回过头去看它的解说。

之后，你将不再惧怕和AIDL相关的面试题。

作者：goeasyway

链接：https://www.jianshu.com/p/467016b4487c

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 你是如何解决Android的布局嵌套问题的？

我们都清楚Android界面的布局太复杂，嵌套层次过深，会使整个界面的测量、布局和绘制变得更复杂，对性能会造成影响。所以我们在写Layout文件时，也要尽量避免布局的嵌套层次过深的问题。

在怎么解决问题之前，我们得有一个好方法先判断当前的问题情况。Android SDK工具箱中有一个叫做Hierarchy Viewer的工具，能够在App运行时分析Layout。

注意：在ROOT的手机，或者是安装开发版的ROM的手机可以直接使用Hierarchy Viewer。

如果没有Root的手机(SDK 4.1及以上)，需要在你的PC端添加一个环境变量“ANDROID\_HVPROTO=ddm”。

Mac系统的配置如下：

保存后运行：source ~/.bash\_profile

最后可以DDMS的Hierarchy Viewer看到：

下面列举一些面试者常使用的方式。

merge

merge标签的作用是合并UI布局，使用该标签能降低UI布局的嵌套层次。

merge标签可用于两种情况：

布局顶结点是FrameLayout且不需要设置background或padding等属性，可以用merge代替，因为Activity内容试图的parent view就是个FrameLayout，所以可以用merge消除只剩一个。

某布局作为子布局被其他布局include时，使用merge当作该布局的顶节点，这样在被引入时顶结点会自动被忽略，而将其子节点全部合并到主布局中。

ViewStub

ViewStub标签引入的布局默认不会inflate，既不会显示也不会占用位置。 ViewStub常用来引入那些默认不会显示，只在特殊情况下显示的布局，如数据加载进度布局、出错提示布局等。

需要在使用时手动inflate:

ViewStub stub = (ViewStub)findViewById(R.id.error\_layout);

errorView = stub.inflate();

errorView.setVisibility(View.VISIBLE);

ViewStub在一定的程度可以起到减少嵌套层次的作用，特别是很多时候我们的程序可能不需要走到ViewStub的界面。

include

将可复用的组件抽取出来并通过include标签使用，但<include>标签能减少布局的层次吗？

我认为不能。include主要解决的是相同布局的复用问题，它并不能减少布局的层次。

用RelativeLayout代替LinearLayout

很多人为了减少布局层次喜欢用RelativeLayout代替LinearLayout，不过可能达到的效果并不会很明显。层次是减少了，但本身RelativeLayout就会比LinearLayout性能差一点。

有一些界面，比如一个图片和一个文本的布局（ListItem常见的布局方式），可以利用TextView有drawableLeft, drawableRight等属性，完全不需要RelativeLayout或者LinearLayout布局。

你不知道的两种新的布局方式

传统的布局方式存在一定的缺陷，如RelativeLayout要两次测量（measure）它的子View才能知道确切的高度；如果LinearLayout布局的子View有设置了layout\_weight，那么它也需要测量两次才能获得布局的高度。

相对于传统的布局方式，Android官方还推出了两种新的布局方式：ConstraintLayout和FlexboxLayout。

ConstraintLayout

ConstraintLayout即约束布局，在2016年由Google I/O推出。ConstraintLayout和RelativeLayout有点类似，控件之间根据依赖关系而存在，但比RelativeLayout更加灵活。创建大型复杂的布局仍然可以使用扁平的层级(不用嵌套View Group)，说的简单些就是，再复杂的界面也可以只有2层层次。

要使用ConstraintLayout需要在build.gradle中添加相关的support库：

compile 'com.android.support.constraint:constraint-layout:1.0.2'

Android Studio 2.3及之后的版本使用引导创建Empty的Activity时，默认就是使用ConstraintLayout布局。关于本面试题的答案，官方其实已经明确给出信号了。

使用ConstraintLayout可以有效的解决布局嵌套过多导致的性能问题，官方也对其渲染性能进行了优化，并且ConstraintLayout支持可视化的方式编写布局。

不过学会熟练使用ConstraintLayout会需要一点时间，但这是值得的。

官方文档：https://developer.android.com/training/constraint-layout/index.html

FlexBoxLayout

做过前端开发（CSS方面）的同学对FlexBox一定不会陌生，最近我在做微信小程序开发时也涉及到FlexBox。FlexBox(弹性布局)是w3c在2009年提出的一种新的布局方案，解决以前那种传统css的盒模型的局限性。

Google开源了FlexboxLayout布局和前端CSS FlexBox布局具有相同的功能（肯定有不一样的地方），但已经足够在Android上改进布局的构建方式。

项目地址：https://github.com/google/flexbox-layout

FlexBoxLayout可以理解成一种更高级的LinearLayout，不过比LinearLayout更加强大和灵活。如果我们使用LinearLayout布局的话，那么不同的分辨率，也许我们要重新调整布局，势必会需要跟多的布局文件放在不同的资源目录。而使用FlexBoxLayout来布局的话，它可以适应各种界面的改变（所以叫响应式布局）。

看一下官方图片感受一下：

如果对前端的Flexbox不太了解的话，你还需要补一些概念，好在这些东西在网上很容易找到。

小结

可能很多读者会觉这样的面试题是吹毛求疵，很多项目中哪有这么复杂的界面，根本就用不到这些优化措施。

So，你没有成长为厉害的人。

可以说厉害的人，或者叫高手，可能只是比较多在意这些细节而已。在实践中的经历告诉我，很多难于解决的性能问题，并不是因为有一个影响性能的问题无法攻克，而是没有一个明显的制约因素，是有各种小问题一点一点堆积起来，最终积重难返。

所以，把细节做好，或者意识到细节的地方可能引发的问题，对我们解决问题是很有帮助的，不要浪费了让你可以成长的细节。

作者：goeasyway

链接：https://www.jianshu.com/p/c8d4a3d7fd26

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 说说Java的内存模型

说实话，把我问的有点“蒙”，确实知道一二，但在工作中很少总结这个方面，以前也专门看过，但那又是太遥远的事情了。硬着头皮把一些想法和记忆说了出来。

有读者会纳闷了，这样的题都能“吓蒙”你？

面试官不一定是最好的面试者，就像教练不一定非要是世界冠军。

面试官的套路和我预想的不一样，没有关注项目经验、管理和架构设计，上来就Java基础到Androd基础，而且极其细致。

Java的内存模型

Java开发人员并不需要像C/C++开发人员，需要时刻注意内存的分配和释放，而是全权交给虚拟机（JVM）去管理，自然关于内存管理或是内存的模型、结构对Java开发来说就是一个“黑箱”。

两眼一抹黑似乎也不影响写Java的代码。但我也说过，了解一些内部的机制或者是自己认为不重要的东西，也许会很有帮助。

最简单的，我们也应该了解Java的堆和栈。而我们所谓的内存管理，基本上指对堆内存的管理，那堆内存在JVM的内存结构中的那个位置呢？

什么是JVM内存

Java源代码文件(.java)会被Java编译器编译为字节码文件(.class)，然后由JVM中的类加载器加载各个类的字节码文件，加载完毕之后，交由JVM执行引擎执行。

JVM在执行Java程序的过程中会把它所管理的内存划分为若干个不同的数据区域。

JVM会用一段空间来存储程序执行期间需要用到的数据和相关信息，这段空间一般被称作为Runtime Data Area（运行时数据区），也就是我们常说的JVM内存。

了解清楚JVM的内存结构会更有助于我们理解Java的内存模型。

我们可以把上图的“运行时数据区”分为线程私有和共享数据区两大类。其中线程私有的数据区包含程序计数器、虚拟机栈、本地方法区，所有线程共享的数据区包含Java堆、方法区，在方法区内有一个常量池。

程序计数器（PC Register）

记录正在执行的虚拟机字节码的地址。和计算机组成原理中提到的程序计数器PC概念类似，是线程私有的，用来记录当前执行的字节码位置。

虚拟机栈（JVM Stack）

也就是我们常常所说的栈。

方法执行的内存区，每个方法执行时会在虚拟机栈中创建栈帧。虚拟机栈的生命周期与线程相同，每个方法（不包含native方法）执行的同时都会创建一个栈帧结构，方法执行过程，对应着虚拟机栈的入栈到出栈的过程。

本地方法栈（Native Method Stack）

本地方法栈则为虚拟机使用到的Native方法提供内存空间。

Java堆（Heap）

Java堆一般是JVM管理的内存中最大的一块，堆在主内存中，是被所有线程共享的一块内存区域，其随着JVM的创建而创建，是用来存储对象本身的以及数组，同时JAVA堆也是GC管理的主要区域。

方法区（Method Area）

主要存放的是已被虚拟机加载的类信息、常量、静态变量、编译器编译后的代码等数据。

常量池（Runtime Constant Pool）

存放编译器生成的各种字面量和符号引用，是方法区的一部分。

内存模型

Java内存模型即Java Memory Model，简称JMM。JMM定义了Java 虚拟机(JVM)在计算机内存(RAM)中的工作方式。Java线程之间的通信由JMM控制，JMM决定一个线程对共享变量的写入何时对另一个线程可见。

从抽象的角度来看，JMM定义了线程和主内存之间的抽象关系：线程之间的共享变量存储在主内存（上面提到的Java堆内存）中，每个线程都有一个私有的本地内存，本地内存中存储了该线程以读/写共享变量的副本。

在命令式编程中，线程之间的通信机制有两种：共享内存和消息传递。

Java内存模型与上面提到的JVM运行时数据区（JVM Runtime Data Areas）两个概念容易混淆。JVM 运行时数据区定义了JVM运行期内存的管理划分，而Java内存模型定义了程序中各个共享变量的访问规则。

自己的面试总结

关于Java的内存模型，我觉得对于Android应用开发比较有益的就是：更容易理解线程安全和并发编程的问题。而后面面试官确实也问到了线程安全，可能这也是一个组合套路吧。

面试完后，虽然有很多题答得都不是很理想，不过对于我这样的“过来人”来说，很清楚面试时你回答的内容并不是最重要的（大多数时候）。重要的是什么？看完这个系列的读者应该心里有数。

附上自己的面试总结：

Java部分准备不充分。

在面试前我对这个职位的信息收集并不充分，我的侧重点在Android的项目框架和技术管理上。但美团一面的面试官视乎是比较重基础知识，而且每个点都问得比较仔细。

没有问面试官的姓名。

下来都不好和面试官做朋友，不是吗？万一以后是同事，还不知道对方是谁也有点尴尬。以前有过，和一个同事处了一段时间了，他才告诉我之前是他面的我。

表达了一些负面信息。

解释了一些不足的地方，个人一直不喜欢“强调”负面信息，在我的一些表达中，还是不自觉的先抑后扬了，不过好在面试官视乎不太在意。

“标准答案”

标准答案为何打引号，请关注Android面试一天一题（Day 43：设计模式）的说明。

面试题：Java的内存模型

标准答案:Java内存模型即Java Memory Model，简称JMM。JMM定义了Java 虚拟机(JVM)在计算机内存(RAM)中的工作方式。程序中的变量存储在主内存中，每个线程拥有自己的工作内存并存放变量的拷贝，线程读写自己的工作内存，通过主内存进行变量的交互。JMM就是规定了工作内存和主内存之间变量访问的细节，通过保障原子性、有序性、可见性来实现线程的有效协同和数据的安全。

面试题：JVM如何判断一个对象实例是否应该被回收？

标准答案: 垃圾回收器会建立有向图的方式进行内存管理，通过GC Roots来往下遍历，当发现有对象处于不可达状态的时候，就会对其标记为不可达，以便于后续的GC回收。

面试题：说说JVM的垃圾回收策略。

标准答案: JVM采用分代垃圾回收。在JVM的内存空间中把堆空间分为年老代和年轻代。将大量创建了没多久就会消亡的对象存储在年轻代，而年老代中存放生命周期长久的实例对象。

作者：goeasyway

链接：https://www.jianshu.com/p/7e0833df599b

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 多个进程同时调用一个ContentProvider的query获取数据，ContentPrvoider是如何反应的呢？

我们知道像Activity这样的组件，它的生命周期回调函数是在UI线程中运行的，ContentProvider的onCreate()也是在UI线程运行，回答这个面试题前，我们要搞清楚ContentProvider的query()，insert()，delete()，update()这几个方法是在UI线程中运行的吗？

如果是，那么就等于说在这里做长时间的操作的话，就会有ANR的问题。如果不是，那是在一个工作线程还是多个线程中呢，即ContentProvider是否支持并发操作？

ContentResolver与ContentProvider类隐藏了实现细节，但是ContentProvider所提供的query()，insert()，delete()，update()都是在ContentProvider进程的线程池中被调用执行的，而不是进程的主线程中。因为那些方法可能同时被多个线程所调用，所以他们都应该是线程安全的。

ContentProvider实现了进程间通信也是基于Binder机制的，所以会回到Binder的线程处理问题。并不是每个ContentProvider都会有一个线程池，而一个进程会共有一个线程池，其实就是Binder线程池。

我之前看到过说明是，每个ContentProvider会有线程池在管理每个客户端ContentResolver的请求，每个线程池有16个线程，不过后来也一直没有在官网上找到资料印证。

实验

当我聊到这个16的数量问题时，面试官有一些疑问。需要我对这个数量的由来做一个说明。

如何证明这个线程池确实是16个线程呢？

有两种方法：

直接查看ContentProvider或者Binder的相关源码证实；

直接写一个应用Demo来证实。

我们来试试第2种方式，现在写一个简单的测试Demo，一个ContentProvider（里面操作数据的地方，我们让它sleep 10秒钟），一个调用者Activity同时发送几十个请求。在ContentProvider的方法中打出Log看运行在哪个线程，以及这几十个请求的执行情况。

ContentProvider代码：

public class MyContentProvider extends ContentProvider {

private final static String TAG = "MyContentProvider";

@Override

public boolean onCreate() {

Log.i(TAG, "onCreate() threadName= " + Thread.currentThread().getName()

+ " threadId=" + Thread.currentThread().getId());

return false;

}

@Override

public Uri insert(Uri uri, ContentValues values) {

Log.i(TAG, "insert() threadName= " + Thread.currentThread().getName()

+ " threadId=" + Thread.currentThread().getId());

try {

Thread.sleep(10000); // 模拟耗时工作

} catch (InterruptedException e) {

e.printStackTrace();

}

return null;

}

......

调用方Activity的代码：

public class MainActivity extends AppCompatActivity {

private final static String TAG = "MainActivity";

public static final Uri CONTENT\_URI = Uri.parse("content://net.goeasyway.myprovider");

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

for (int i = 0; i < 50; i++) {

new Thread(new Runnable() {

@Override

public void run() {

getContentResolver().insert(CONTENT\_URI, new ContentValues());

}

}).start();

}

}

}

测试结果分两种情况：

1、同一进程

ContentProvider和调用getContentResolver()在同一进程中的情况，即AndroidManifest.xml这样配置provider标签的：

<provider

android:authorities="net.goeasyway.myprovider"

android:name=".MyContentProvider" />

测试结果发现，ContentProvider.insert()会生成一个新的线程去处理请求，所以getContentResolver().insert()请求了50次，就会ContentProvider的进程中产生50个新的线程去处理insert()操作。

I/MyContentProvider: insert() threadName= Thread-1 threadId=1304

I/MyContentProvider: insert() threadName= Thread-3 threadId=1306

I/MyContentProvider: insert() threadName= Thread-2 threadId=1305

I/MyContentProvider: insert() threadName= Thread-4 threadId=1307

I/MyContentProvider: insert() threadName= Thread-5 threadId=1308

I/MyContentProvider: insert() threadName= Thread-6 threadId=1309

I/MyContentProvider: insert() threadName= Thread-7 threadId=1310

I/MyContentProvider: insert() threadName= Thread-8 threadId=1311

I/MyContentProvider: insert() threadName= Thread-9 threadId=1312

......

I/MyContentProvider: insert() threadName= Thread-47 threadId=1350

I/MyContentProvider: insert() threadName= Thread-49 threadId=1352

I/MyContentProvider: insert() threadName= Thread-48 threadId=1351

I/MyContentProvider: insert() threadName= Thread-50 threadId=1353

2、不同进程

ContentProvider和调用getContentResolver()不在同一进程中的情况，配置provider标签时给它调置一个远程进程：

<provider

android:authorities="net.goeasyway.myprovider"

android:name=".MyContentProvider"

android:process=":remote"/>

这样配置MyContentProvider将运行在“包名:remote”的进程中，和MainActivity所在进程“包名”之间进程间通信。测试的结果发现MyContentProvider中的insert()方法运行在Binder线程池的线程中。

I/MyContentProvider: insert() threadName= Binder:10436\_1 threadId=1285

I/MyContentProvider: insert() threadName= Binder:10436\_2 threadId=1286

I/MyContentProvider: insert() threadName= Binder:10436\_3 threadId=1288

I/MyContentProvider: insert() threadName= Binder:10436\_4 threadId=1289

I/MyContentProvider: insert() threadName= Binder:10436\_5 threadId=1290

I/MyContentProvider: insert() threadName= Binder:10436\_6 threadId=1291

I/MyContentProvider: insert() threadName= Binder:10436\_7 threadId=1292

I/MyContentProvider: insert() threadName= Binder:10436\_8 threadId=1293

I/MyContentProvider: insert() threadName= Binder:10436\_9 threadId=1294

I/MyContentProvider: insert() threadName= Binder:10436\_A threadId=1295

I/MyContentProvider: insert() threadName= Binder:10436\_B threadId=1296

I/MyContentProvider: insert() threadName= Binder:10436\_C threadId=1297

I/MyContentProvider: insert() threadName= Binder:10436\_D threadId=1298

I/MyContentProvider: insert() threadName= Binder:10436\_E threadId=1299

I/MyContentProvider: insert() threadName= Binder:10436\_F threadId=1300

I/MyContentProvider: insert() threadName= Binder:10436\_10 threadId=1301

......

每次MyContentProvider的insert()只能在16个Binder线程中执行，只有等这16个线程有空闲之后才后处理剩下的getContentResolver().insert()的请求。

笔者经验分享：虽然去研究ContentProvider的启动和调用流程代码肯定也能得出上面的结论，但还是会有一些难度。所以，工作中有时候想太多还不如直接动手试一下，结论一下子就出来了。

和面试官的“争论”

这个结论是不是有问题呢？

当我聊到这个线程池中有16个线程可供使用时，面试官对这个结论提出了一些看法。一个进程创建后为了和在系统进程中的AMS（ActivityManangerService）通信，一个APK进程本身就是在Binder线程池中占用2个线程来维持Binder的通信。

那么，上面提到的第二种不同进程的案例中，ContentProvider能同时使用的线程数量应该是14个才对啊。怎么我们测试的结果会是16呢？

这一点我当时和面试官是有分歧的。因为我记得做过实验结果就是16个线程，但如过一个进程只维护一个Binder线程池的话，确实应该是14个线程才对。

在这点上，我确实没有理解清楚Binder的线程池，所以我知道了实际效果可以同时运行16个线程（假设自己的实验没出错），但却无法解释面试官提的“为什么不是14个的问题”。

我面试回来后，看一些说明和代码片段，一个初步的判断是，面试官提到的和AMS通信用到的线程并不会算在这个线程池中。

欢迎提出异议或者你认为正确的解释。

高级中的高级

可能很多读者会觉得面试官这样问有点吹毛求疵了，这不是为难人吗？

锤子科技把高级工程师还进行了细分，即高级中再来一个“初级、中级、高级”这类的级别，很多公司应该也有类似定级别的（比如有喜欢P的、有喜欢T的）。高级中的高级和高级中的初级，你觉得会差在哪里呢？

所以，作为面试者你应该仔细思考一下，很多时候面试官过问的细节往往就是差距所在。

“标准答案”

面试题：多个进程同时调用一个ContentProvider的query获取数据，ContentPrvoider是如何反应的呢？

标准答案：一个content provider可以接受来自另外一个进程的数据请求。尽管ContentResolver与ContentProvider类隐藏了实现细节，但是ContentProvider所提供的query()，insert()，delete()，update()都是在ContentProvider进程的线程池中被调用执行的，而不是进程的主线程中。这个线程池是有Binder创建和维护的，其实使用的就是每个应用进程中的Binder线程池。

面试题：你觉得Android设计ContentProvider的目的是什么呢？

标准答案：1. 隐藏数据的实现方式，对外提供统一的数据访问接口；

2.更好的数据访问权限管理。ContentProvider可以对开发的数据进行权限设置，不同的URI可以对应不同的权限，只有符合权限要求的组件才能访问到ContentProvider的具体操作。

3.ContentProvider封装了跨进程共享的逻辑，我们只需要Uri即可访问数据。由系统来管理ContentProvider的创建、生命周期及访问的线程分配，简化我们在应用间共享数据（进程间通信）的方式。我们只管通过ContentResolver访问ContentProvider所提示的数据接口，而不需要担心它所在进程是启动还是未启动。

面试题：运行在主线程的ContentProvider为什么不会影响主线程的UI操作?

标准答案：

ContentProvider的onCreate()是运行在UI线程的，而query()，insert()，delete()，update()是运行在线程池中的工作线程的，所以调用这向个方法并不会阻塞ContentProvider所在进程的主线程，但可能会阻塞调用者所在的进程的UI线程！

所以，调用ContentProvider的操作仍然要放在子线程中去做。虽然直接的CRUD的操作是在工作线程的，但系统会让你的调用线程等待这个异步的操作完成，你才可以继续线程之前的工作。

作者：goeasyway

链接：https://www.jianshu.com/p/c70ae80cf64d

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 如何导入外部数据库

把原数据库包括在项目源码的 res/raw

android系统下数据库应该存放在 /data/data/com..（package name）/ 目录下，所以我们需要做的是把已有的数据库传入那个目录下.操作方法是用FileInputStream读取原数据库，再用FileOutputStream把读取到的东西写入到那个目录.

## 本地广播和全局广播有什么差别

因广播数据在本应用范围内传播，不用担心隐私数据泄露的问题。 不用担心别的应用伪造广播，造成安全隐患。 相比在系统内发送全局广播，它更高效。

## intentService作用是什么,AIDL解决了什么问题-小米

android跨进程通信（IPC）：使用AIDL

生成一个默认的且与主线程互相独立的工作者线程来执行所有传送至onStartCommand() 方法的Intetnt。

生成一个工作队列来传送Intent对象给你的onHandleIntent()方法，同一时刻只传送一个Intent对象，这样一来，你就不必担心多线程的问题。在所有的请求(Intent)都被执行完以后会自动停止服务，所以，你不需要自己去调用stopSelf()方法来停止。

该服务提供了一个onBind()方法的默认实现，它返回null

提供了一个onStartCommand()方法的默认实现，它将Intent先传送至工作队列，然后从工作队列中每次取出一个传送至onHandleIntent()方法，在该方法中对Intent对相应的处理。

AIDL (Android Interface Definition Language) 是一种IDL 语言，用于生成可以在Android设备上两个进程之间进行进程间通信(interprocess communication, IPC)的代码。如果在一个进程中（例如Activity）要调用另一个进程中（例如Service）对象的操作，就可以使用AIDL生成可序列化的参数。 AIDL IPC机制是面向接口的，像COM或Corba一样，但是更加轻量级。它是使用代理类在客户端和实现端传递数据。

## Activity Window View三者的差别,fragment的特点-360

Activity像一个工匠（控制单元），Window像窗户（承载模型），View像窗花（显示视图） LayoutInflater像剪刀，Xml配置像窗花图纸。

在Activity中调用attach，创建了一个Window

创建的window是其子类PhoneWindow，在attach中创建PhoneWindow

在Activity中调用setContentView(R.layout.xxx)

其中实际上是调用的getWindow().setContentView()

调用PhoneWindow中的setContentView方法

创建ParentView：作为ViewGroup的子类，实际是创建的DecorView(作为FramLayout的子类）

将指定的R.layout.xxx进行填充，通过布局填充器进行填充【其中的parent指的就是DecorView】

调用到ViewGroup

调用ViewGroup的removeAllView()，先将所有的view移除掉

添加新的view：addView()

Fragment 特点

Fragment可以作为Activity界面的一部分组成出现；

可以在一个Activity中同时出现多个Fragment，并且一个Fragment也可以在多个Activity中使用；

在Activity运行过程中，可以添加、移除或者替换Fragment；

Fragment可以响应自己的输入事件，并且有自己的生命周期，它们的生命周期会受宿主Activity的生命周期影响。

## Handler,Thread和HandlerThread的差别-小米

探索 Android 大杀器—— Handler

Android异步消息处理机制完全解析，带你从源码的角度彻底理解

详解 Android 中的 HandlerThread

Android应用程序消息处理机制（Looper、Handler）分析

从Android中Thread（java.lang.Thread -> java.lang.Object）描述可以看出，Android的Thread没有对Java的Thread做任何封装，但是Android提供了一个继承自Thread的类HandlerThread（android.os.HandlerThread -> java.lang.Thread），这个类对Java的Thread做了很多便利Android系统的封装。

android.os.Handler可以通过Looper对象实例化，并运行于另外的线程中，Android提供了让Handler运行于其它线程的线程实现，也是就HandlerThread。HandlerThread对象start后可以获得其Looper对象，并且使用这个Looper对象实例Handler。

## LaunchMode应用场景-百度-小米-乐视

standard，创建一个新的Activity。

singleTop，栈顶不是该类型的Activity，创建一个新的Activity。否则，onNewIntent。

singleTask，回退栈中没有该类型的Activity，创建Activity，否则，onNewIntent+ClearTop。

注意:

设置了"singleTask"启动模式的Activity，它在启动的时候，会先在系统中查找属性值affinity等于它的属性值taskAffinity的Task存在； 如果存在这样的Task，它就会在这个Task中启动，否则就会在新的任务栈中启动。因此， 如果我们想要设置了"singleTask"启动模式的Activity在新的任务中启动，就要为它设置一个独立的taskAffinity属性值。

如果设置了"singleTask"启动模式的Activity不是在新的任务中启动时，它会在已有的任务中查看是否已经存在相应的Activity实例， 如果存在，就会把位于这个Activity实例上面的Activity全部结束掉，即最终这个Activity 实例会位于任务的Stack顶端中。

在一个任务栈中只有一个”singleTask”启动模式的Activity存在。他的上面可以有其他的Activity。这点与singleInstance是有区别的。

singleInstance，回退栈中，只有这一个Activity，没有其他Activity。

singleTop适合接收通知启动的内容显示页面。

例如，某个新闻客户端的新闻内容页面，如果收到10个新闻推送，每次都打开一个新闻内容页面是很烦人的。

singleTask适合作为程序入口点。

例如浏览器的主界面。不管从多少个应用启动浏览器，只会启动主界面一次，其余情况都会走onNewIntent，并且会清空主界面上面的其他页面。

singleInstance应用场景：

闹铃的响铃界面。 你以前设置了一个闹铃：上午6点。在上午5点58分，你启动了闹铃设置界面，并按 Home 键回桌面；在上午5点59分时，你在微信和朋友聊天；在6点时，闹铃响了，并且弹出了一个对话框形式的 Activity(名为 AlarmAlertActivity) 提示你到6点了(这个 Activity 就是以 SingleInstance 加载模式打开的)，你按返回键，回到的是微信的聊天界面，这是因为 AlarmAlertActivity 所在的 Task 的栈只有他一个元素， 因此退出之后这个 Task 的栈空了。如果是以 SingleTask 打开 AlarmAlertActivity，那么当闹铃响了的时候，按返回键应该进入闹铃设置界面。

## 什么情况导致内存泄漏-美团

1.资源对象没关闭造成的内存泄漏

描述： 资源性对象比如(Cursor，File文件等)往往都用了一些缓冲，我们在不使用的时候，应该及时关闭它们，以便它们的缓冲及时回收内存。它们的缓冲不仅存在于 java虚拟机内，还存在于java虚拟机外。如果我们仅仅是把它的引用设置为null,而不关闭它们，往往会造成内存泄漏。因为有些资源性对象，比如 SQLiteCursor(在析构函数finalize(),如果我们没有关闭它，它自己会调close()关闭)，如果我们没有关闭它，系统在回收它时也会关闭它，但是这样的效率太低了。因此对于资源性对象在不使用的时候，应该调用它的close()函数，将其关闭掉，然后才置为null.在我们的程序退出时一定要确保我们的资源性对象已经关闭。 程序中经常会进行查询数据库的操作，但是经常会有使用完毕Cursor后没有关闭的情况。如果我们的查询结果集比较小，对内存的消耗不容易被发现，只有在常时间大量操作的情况下才会复现内存问题，这样就会给以后的测试和问题排查带来困难和风险。

2.构造Adapter时，没有使用缓存的convertView

描述： 以构造ListView的BaseAdapter为例，在BaseAdapter中提供了方法： public View getView(int position, ViewconvertView, ViewGroup parent) 来向ListView提供每一个item所需要的view对象。初始时ListView会从BaseAdapter中根据当前的屏幕布局实例化一定数量的 view对象，同时ListView会将这些view对象缓存起来。当向上滚动ListView时，原先位于最上面的list item的view对象会被回收，然后被用来构造新出现的最下面的list item。这个构造过程就是由getView()方法完成的，getView()的第二个形参View convertView就是被缓存起来的list item的view对象(初始化时缓存中没有view对象则convertView是null)。由此可以看出，如果我们不去使用 convertView，而是每次都在getView()中重新实例化一个View对象的话，即浪费资源也浪费时间，也会使得内存占用越来越大。 ListView回收list item的view对象的过程可以查看: android.widget.AbsListView.java --> voidaddScrapView(View scrap) 方法。 示例代码：

public View getView(int position, ViewconvertView, ViewGroup parent) {

View view = new Xxx(...);

... ...

return view;

}

修正示例代码：

public View getView(int position, ViewconvertView, ViewGroup parent) {

View view = null;

if (convertView != null) {

view = convertView;

populate(view, getItem(position));

...

} else {

view = new Xxx(...);

...

}

return view;

}

3.Bitmap对象不在使用时调用recycle()释放内存

描述： 有时我们会手工的操作Bitmap对象，如果一个Bitmap对象比较占内存，当它不在被使用的时候，可以调用Bitmap.recycle()方法回收此对象的像素所占用的内存，但这不是必须的，视情况而定。可以看一下代码中的注释：

/\* •Free up the memory associated with thisbitmap's pixels, and mark the •bitmap as "dead", meaning itwill throw an exception if getPixels() or •setPixels() is called, and will drawnothing. This operation cannot be •reversed, so it should only be called ifyou are sure there are no •further uses for the bitmap. This is anadvanced call, and normally need •not be called, since the normal GCprocess will free up this memory when •there are no more references to thisbitmap. /

4.试着使用关于application的context来替代和activity相关的context

这是一个很隐晦的内存泄漏的情况。有一种简单的方法来避免context相关的内存泄漏。最显著地一个是避免context逃出他自己的范围之外。使用Application context。这个context的生存周期和你的应用的生存周期一样长，而不是取决于activity的生存周期。如果你想保持一个长期生存的对象，并且这个对象需要一个context,记得使用application对象。你可以通过调用 Context.getApplicationContext() or Activity.getApplication()来获得。更多的请看这篇文章如何避免 Android内存泄漏。

5.注册没取消造成的内存泄漏

一些Android程序可能引用我们的Anroid程序的对象(比如注册机制)。即使我们的Android程序已经结束了，但是别的引用程序仍然还有对我们的Android程序的某个对象的引用，泄漏的内存依然不能被垃圾回收。调用registerReceiver后未调用unregisterReceiver。 比如:假设我们希望在锁屏界面(LockScreen)中，监听系统中的电话服务以获取一些信息(如信号强度等)，则可以在LockScreen中定义一个 PhoneStateListener的对象，同时将它注册到TelephonyManager服务中。对于LockScreen对象，当需要显示锁屏界面的时候就会创建一个LockScreen对象，而当锁屏界面消失的时候LockScreen对象就会被释放掉。 但是如果在释放 LockScreen对象的时候忘记取消我们之前注册的PhoneStateListener对象，则会导致LockScreen无法被垃圾回收。如果不断的使锁屏界面显示和消失，则最终会由于大量的LockScreen对象没有办法被回收而引起OutOfMemory,使得system\_process 进程挂掉。 虽然有些系统程序，它本身好像是可以自动取消注册的(当然不及时)，但是我们还是应该在我们的程序中明确的取消注册，程序结束时应该把所有的注册都取消掉。

6.集合中对象没清理造成的内存泄漏

我们通常把一些对象的引用加入到了集合中，当我们不需要该对象时，并没有把它的引用从集合中清理掉，这样这个集合就会越来越大。如果这个集合是static的话，那情况就更严重了。

## Service与Activity之间通信的几种方式

通过Binder对象

通过broadcast(广播)的形式

## 如何保证service在后台不被Kill

一、onStartCommand方法，返回START\_STICKY

START\_STICKY 在运行onStartCommand后service进程被kill后，那将保留在开始状态，但是不保留那些传入的intent。不久后service就会再次尝试重新创建，因为保留在开始状态，在创建 service后将保证调用onstartCommand。如果没有传递任何开始命令给service，那将获取到null的intent。

START\_NOT\_STICKY 在运行onStartCommand后service进程被kill后，并且没有新的intent传递给它。Service将移出开始状态，并且直到新的明显的方法（startService）调用才重新创建。因为如果没有传递任何未决定的intent那么service是不会启动，也就是期间onstartCommand不会接收到任何null的intent。

START\_REDELIVER\_INTENT 在运行onStartCommand后service进程被kill后，系统将会再次启动service，并传入最后一个intent给onstartCommand。直到调用stopSelf(int)才停止传递intent。如果在被kill后还有未处理好的intent，那被kill后服务还是会自动启动。因此onstartCommand不会接收到任何null的intent。

二、提升service进程优先级

Android中的进程是托管的，当系统进程空间紧张的时候，会依照优先级自动进行进程的回收。Android将进程分为6个等级,它们按优先级顺序由高到低依次是:

前台进程( FOREGROUND\_APP)

可视进程(VISIBLE\_APP )

次要服务进程(SECONDARY\_SERVER )

后台进程 (HIDDEN\_APP)

内容供应节点(CONTENT\_PROVIDER)

空进程(EMPTY\_APP)

当service运行在低内存的环境时，将会kill掉一些存在的进程。因此进程的优先级将会很重要，可以使用startForeground 将service放到前台状态。这样在低内存时被kill的几率会低一些。

三、onDestroy方法里重启service

service +broadcast 方式，就是当service走ondestory的时候，发送一个自定义的广播，当收到广播的时候，重新启动service；

四、Application加上Persistent属性

五、监听系统广播判断Service状态

通过系统的一些广播，比如：手机重启、界面唤醒、应用状态改变等等监听并捕获到，然后判断我们的Service是否还存活，别忘记加权限啊。

## Requestlayout,onlayout,onDraw,DrawChild区别与联系-猎豹

requestLayout()方法 ：会导致调用measure()过程 和 layout()过程 。 将会根据标志位判断是否需要ondraw

onLayout()方法(如果该View是ViewGroup对象，需要实现该方法，对每个子视图进行布局)

调用onDraw()方法绘制视图本身 (每个View都需要重载该方法，ViewGroup不需要实现该方法)

drawChild()去重新回调每个子视图的draw()方法

## LinearLayout和RelativeLayout性能对比-百度

RelativeLayout会让子View调用2次onMeasure，LinearLayout 在有weight时，也会调用子View2次onMeasure

RelativeLayout的子View如果高度和RelativeLayout不同，则会引发效率问题，当子View很复杂时，这个问题会更加严重。如果可以，尽量使用padding代替margin。

在不影响层级深度的情况下,使用LinearLayout和FrameLayout而不是RelativeLayout。

最后再思考一下文章开头那个矛盾的问题，为什么Google给开发者默认新建了个RelativeLayout，而自己却在DecorView中用了个LinearLayout。因为DecorView的层级深度是已知而且固定的，上面一个标题栏，下面一个内容栏。采用RelativeLayout并不会降低层级深度，所以此时在根节点上用LinearLayout是效率最高的。而之所以给开发者默认新建了个RelativeLayout是希望开发者能采用尽量少的View层级来表达布局以实现性能最优，因为复杂的View嵌套对性能的影响会更大一些。

## 优化自定义view百度-乐视-小米

为了加速你的view，对于频繁调用的方法，需要尽量减少不必要的代码。先从onDraw开始，需要特别注意不应该在这里做内存分配的事情，因为它会导致GC，从而导致卡顿。在初始化或者动画间隙期间做分配内存的动作。不要在动画正在执行的时候做内存分配的事情。

你还需要尽可能的减少onDraw被调用的次数，大多数时候导致onDraw都是因为调用了invalidate().因此请尽量减少调用invaildate()的次数。如果可能的话，尽量调用含有4个参数的invalidate()方法而不是没有参数的invalidate()。没有参数的invalidate会强制重绘整个view。

另外一个非常耗时的操作是请求layout。任何时候执行requestLayout()，会使得Android UI系统去遍历整个View的层级来计算出每一个view的大小。如果找到有冲突的值，它会需要重新计算好几次。另外需要尽量保持View的层级是扁平化的，这样对提高效率很有帮助。

如果你有一个复杂的UI，你应该考虑写一个自定义的ViewGroup来执行他的layout操作。与内置的view不同，自定义的view可以使得程序仅仅测量这一部分，这避免了遍历整个view的层级结构来计算大小。这个PieChart 例子展示了如何继承ViewGroup作为自定义view的一部分。PieChart 有子views，但是它从来不测量它们。而是根据他自身的layout法则，直接设置它们的大小。